UI++

Version 2.10.4.0

April 29, 2018

Jason Sandys

https://home.configmgrftw.com

1 CONTENTS

2 WHAT IS IT

UI++ is a better way to display information to the interactive user, solicit input from that same interactive user, and populate task sequences variables during (surprise) System Center Configuration Manager (ConfigMgr) Operating System Deployment (OSD). UI++ is completely redesigned from OSD++ version 1 and also rolls in the features of OSD AppTree.

UI++ is completely customizable: each and every message, prompt, hint, value, etc. is controllable using an easy to create XML based configuration file.

In addition to simply displaying information to the user and prompting them for input, UI++ can also read from and write to the registry, query and write to WMI, and display a tree of packages and applications to choose for installation in the task sequence.

Although designed with ConfigMgr OSD in mind, there are potentially many other uses for UI++.

3 WHERE CAN I GET IT

UI++ is available at https://home.configmgrftw.com/uiplusplus/.

Please send all bugs, features requests, comments, etc. to me using the contact form there.

4 CURRENT CHANGE LOG

The following is the current change log for changes since version 2.10.0.0. For changes prior to that, see section 11 ("Change Log History").

All items listed below are hyperlinked to their corresponding sections in the documentation for quick access where applicable.

2.10.4.0

- Fixed
  - **UserAuth** action with the **GetGroups** attribute set to True successfully checked authentication but did not allow progress to the next action.
  - Condition evaluation bug where in some cases conditions based on single variable values did not correctly return True.
- Updated
  - Regular Expression matches on **Case** elements within a **Switch** action now match on substrings instead of just the complete string specified in the **OnValue** attribute.
  - Reverted variable substitution behavior so that if a variable is not found, it is replaced by a blank value.
  - Variable substitution now replaces variables with environment variable values as well as task sequence variable values. Environment variable values take precedence in cases of name collisions.

- **DefaultValues** action determination of the **XSystemUEFI** variable's value now matches what ConfigMgr does for _SMSTSBootUEFI.
- **InputInfo** elements within a **UserInput** action now scale their font size based on the line length to accommodate longer strings.
- The list box portion of a **ChoiceInput** now dynamically increases its width to show longer choices.
- The edit box portion of a **ChoiceInput** now displays a tool tip if the user hovers over it and the choice selected is too long.

2.10.3.0

- Added
  - Ability to set UI++ windows to not always on top.
  - Ability to specify additional user attributes to collect during **UserAuth** and populate into variables.
  - New **SaveItems** action type to copy or save files and debug information to a specific location.
- Updated
  - **UserAuth** behavior when user is not a member of the groups specified. Instead of failing completely, this now simply counts as a single failure allowing the interactive user to try again up to the defined **MaxRetryCount**.
  - **UserAuth** optimization.
  - Variable substitution behavior when a specified variable does not exist. Previously, this replaced the variable with an empty string; now, the variable is left intact including the surrounding percent symbols.
- Fixed
  - Multiple bugs when hiding the **DefaultValues** progress bar including it not collecting any values at all and a memory leak.

2.10.2.0

- Added
  - Conditions now work on **Case** and **Variable** elements within a **Switch** action.
  - Ability to force test in a TextInput item to either upper of lower case at input time.
  - Ability to set default values of **ChoiceInput** items based on alternate values.
  - Ability to perform case insensitive regular expression matches in **Switch** actions.
- Updated
  - Performance improvement during **DefaultValues** action.
  - Executables are now signed.

2.10.1.0

- Added
  - Ability to load extension DLLs that contain additional actions.
  - Ability to specify the size of the drop list for a ChoiceInput.
- Fixed
  - Initial field focus bug in the **UserAuth** action.

- Keyboard usability bug in the **UserAuth** action when using a drop-down for domains.

2.10.0.0

- Added
  - ActionGroup element to group Action elements in the configuration XML.
  - Additional detections to the **DefaultValues** action:
    - Is TPM enabled.
    - Is system joined to domain.
    - User principal name.
    - BitLocker protection status of the system drive.
    - OS System Drive.
    - Firewall Information.
    - Management Information.
    - Windows Update Information.
    - Windows Defender Information.
    - Azure AD Information.
  - Added VM and Security value type groups to the **DefaultValues** action.
- Fixed
  - Bug in unsorted **ChoiceInput** fields where the default value was not properly selected at initiation time if the option didn't match the value and the value was specified as the default. This also affected going back to a dialog where a choice was selected on an unsorted **ChoiceInput** where the option didn't match the value.
  - Memory leak when a parsing error was encountered during VBScript expression evaluation.
  - TPM detection.
  - Handling of boolean properties in WMI; instead of returning -1 and 0, True and False are now properly returned.

5 WHAT UI++ DOES

UI++ does whatever you want it to do. Ultimately, its main purpose in life is to gather info from a system or from an interactive user and based upon this info, write values to task sequence variables, the registry, or WMI. Writing the values to task sequence variables allows those values to be consumed later in the task sequence for things like naming the system being deployed, setting the time zone, or installing software. Writing the values to the registry or WMI allows those values to be consumed outside of a task sequence including by hardware inventory.

UI++ can be called like any other executable inside of a task sequence or within Windows. In addition to simply calling the executable, you must supply a configuration file. This file is an XML file that defines all behavior of UI++. This behavior is defined as a series of Actions that include any of the following in any order including multiple occurrences as necessary:

- Showing a Dialog
  - Error Info
  - User Info
  - User Input
  - Application Select Tree
  - Active Directory Authentication
- Creating default task sequence variables based on the current system state
- Reading from the registry
- Writing to the registry
- Querying WMI
- Creating a WMI namespace, class, and/or object
- Setting a task sequence variable

UI++ was mainly designed to be used with ConfigMgr 2012 SP1/R2 and all testing has been done with 2012 R2. It should work fine with ConfigMgr 2007, but I won't guarantee anything there. Additionally, UI++ should work on any supported version and edition of Windows for use outside of a task sequence.

5.1 DIALOGS

All user interface dialogs have the same look and feel including the following:

1. Rounded corners
2. Flat look and feel
3. Major title
4. Dialog title

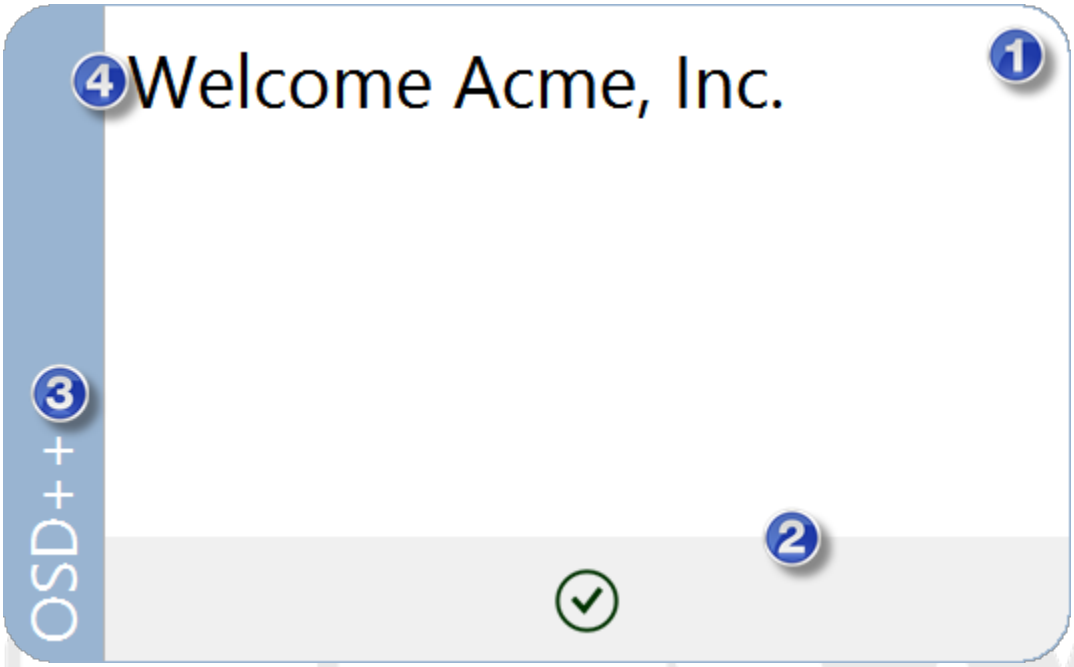


Figure 1: A Base Dialog

5.1.1 Info

The first type of dialog is the simple Info Dialog. This dialog is meant to convey information to the user. It includes an open text area which can be formatted using standard HTML.

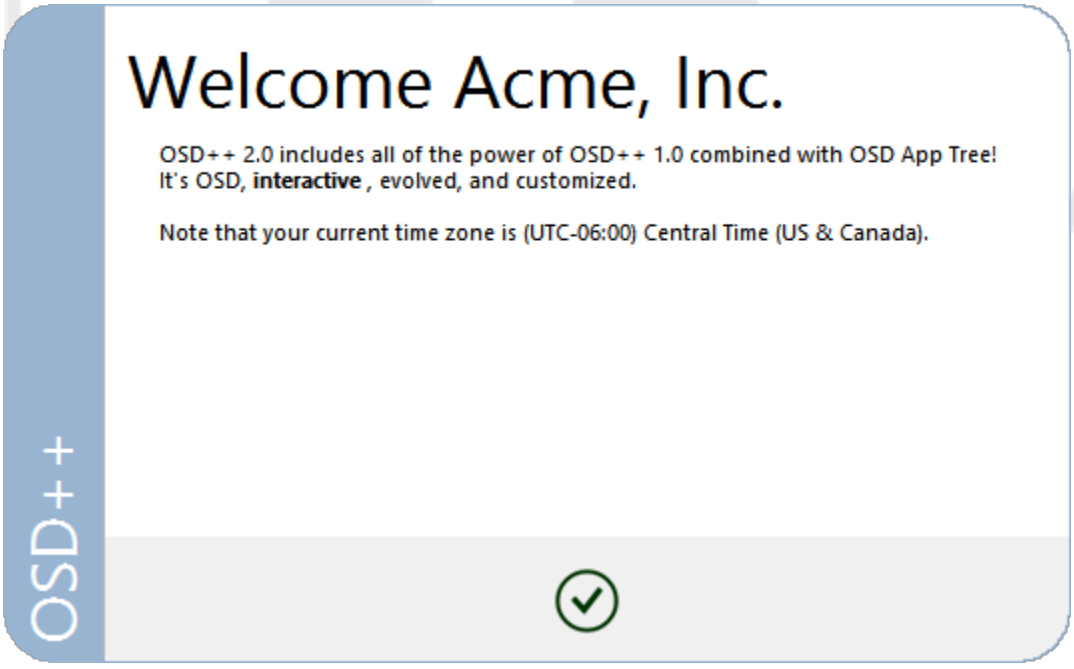


Figure 2: An Info Dialog

5.1.2 Error Info

The **ErrorInfo** dialog is nearly identical to an Info dialog except that it is can only show a **Cancel** button.

5.1.3 Input

User input can take the form of one or more dialog boxes each containing multiple text input boxes, drop-down list boxes, checkboxes, or simply information items.

Each text input box has the following characteristics:

1. A prompt/question
2. A tooltip with a message describing the correct format of the text
3. An exclamation icon indicating that the field is not in the correct format
4. Textboxes are also highlighted in red when they do not have focus and do not contain text matching the correct format
5. A hint is displayed in fields with no text helping the user know what to enter
6. Text within a field that does not match the correct format is colored red.
7. (not shown) A regular expression defining the exact format of acceptable text

Each drop-down list box has the following characteristics:

1. A prompt/question
2. An exclamation icon indicating that a choice has been made or not
3. A drop-down list of customizable choices

Each checkbox item has the following characteristics:

1. A prompt/question

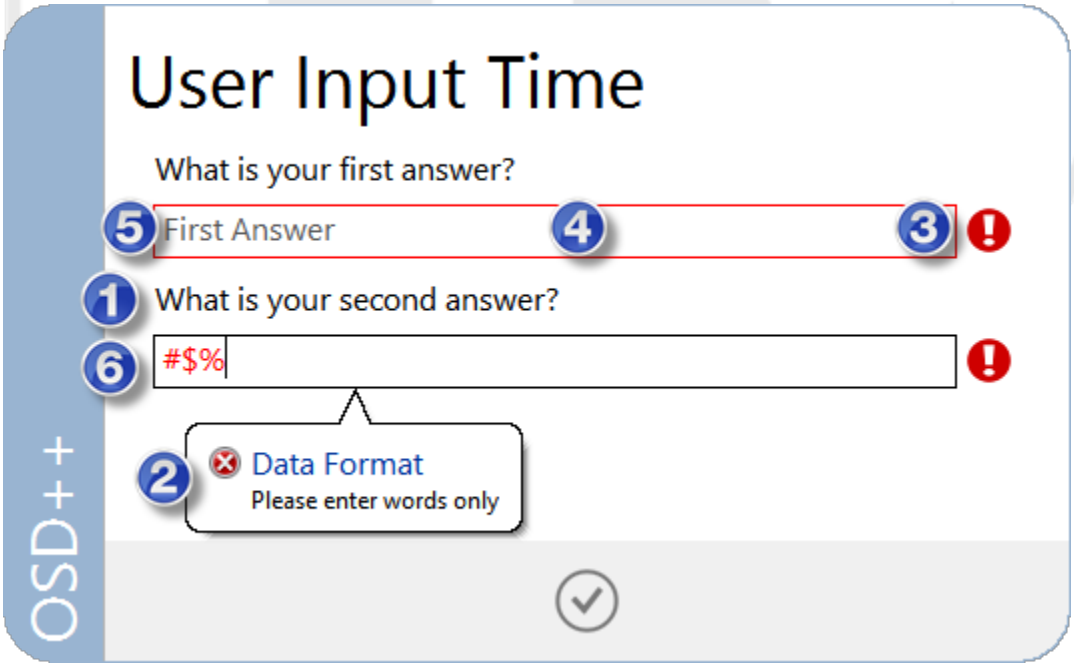


Figure 3: An Input Dialog

If you put in more than fields in an input dialog than will fit, only the first will be shown.

The OK button will remain disabled until all input boxes have matched their defined regular expressions and all drop-down list boxes have a valid selection.

UI++ is not limited to a single dialog box. Multiple dialog boxes can be presented. This may be useful to group specific types of requests together or to prompt for a piece of information, retrieve a value from WMI or the registry using the user's input, and then further prompt the user based on the information from WMI or the registry.

5.1.4 AppTree

An **AppTree** dialog displays a dialog with a tree containing applications and packages for the interactive user to choose from. These applications and packages can be arbitrarily grouped to make selection easier. Applications, packages, and groups can all be set to either selected by default or required. Applications and packages can also reference other applications and packages to automatically include and can also be hidden from the AppTree. Hidden application and packages are not directly selectable in the tree (because they're hidden of course), but can be set to required or default or can be selected because they are specified as included when an application or package that is selected by the user.

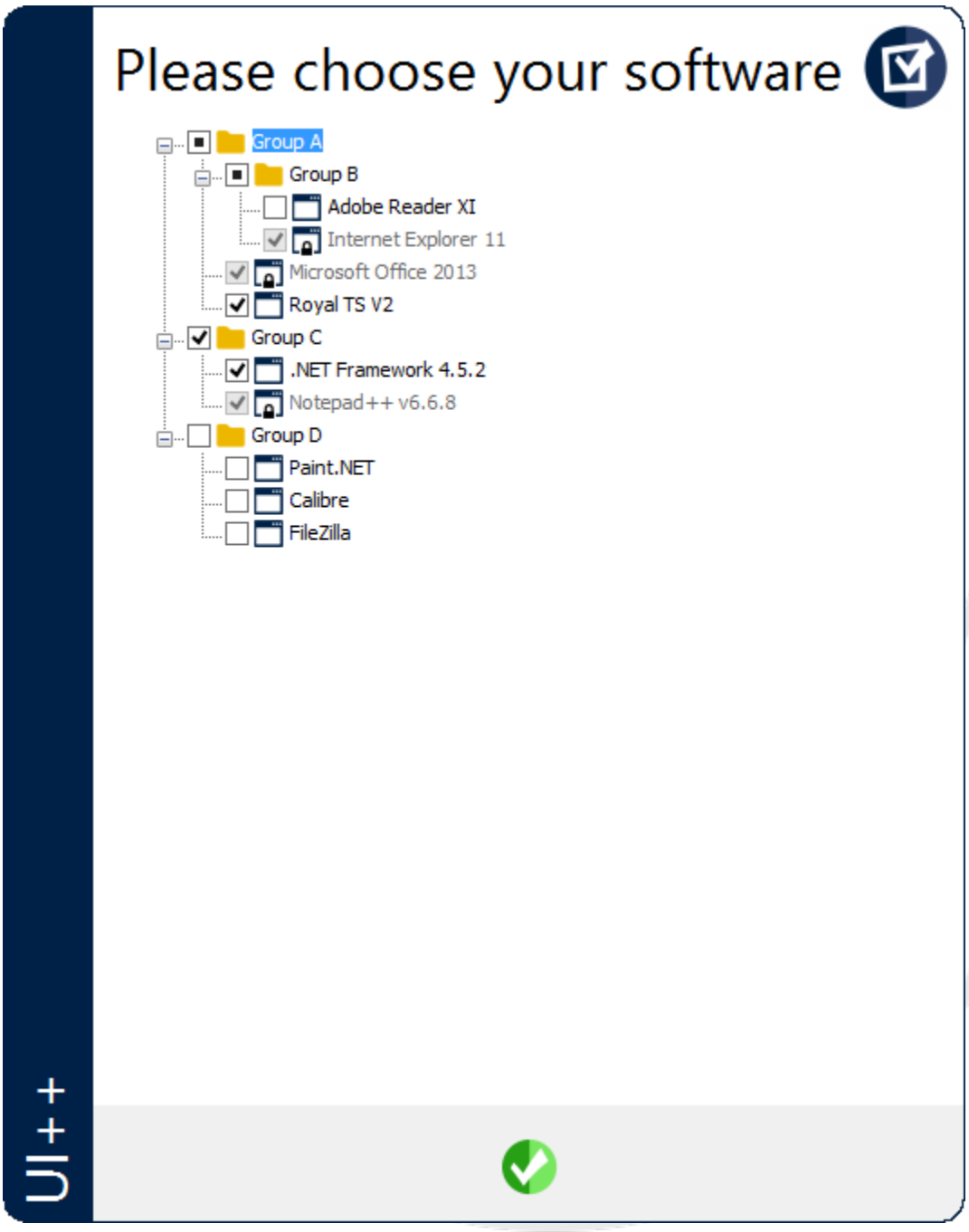


Figure 4: An AppTree Dialog

Applications and packages chosen in an **AppTree** dialog are added to two sets of task sequence variables: one for applications and one for packages. Each set of task sequence variables has a base variable. This base variable should be added to either an Install Application step or Install Software step within the task sequence. These steps with then perform the magic of the installing the chosen applications and packages.

5.1.5 Preflight

This action performs a series of checks on the local system. If all of the checks pass, then the preflight stage is successful and the dialog can be dismissed continuing with additional UI++ actions. If any of checks do not pass however, the dismissing the dialog also stops UI++ and returns an error code to the calling task sequence or process.

The dialog itself displays each check and the result of the check. Checks can be any valid condition as described in section 9.4 (“Conditions”). Up to six checks can be performed and displayed in a normal Preflight dialog and twelve for a tall Preflight dialog.

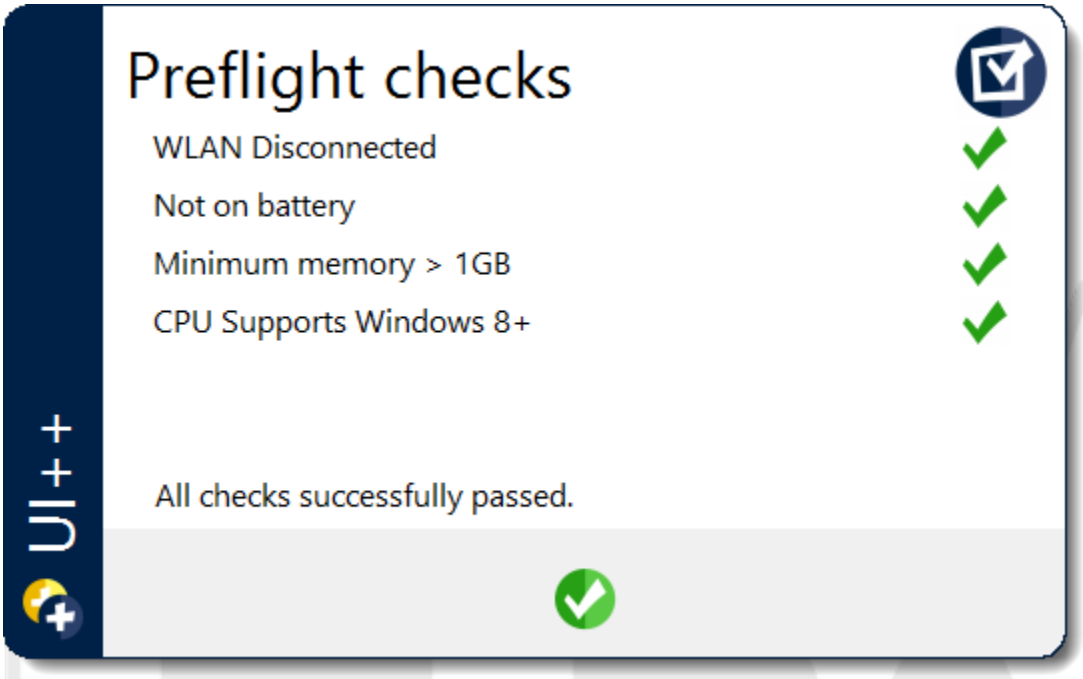


Figure 5: A Preflight Dialog

5.1.6 AD Authentication

This dialog type displays a fixed set of three fields to the interactive user where they must supply their domain credentials in order to authenticate and proceed with the rest of the actions defined for UI++ and the task sequence in general.

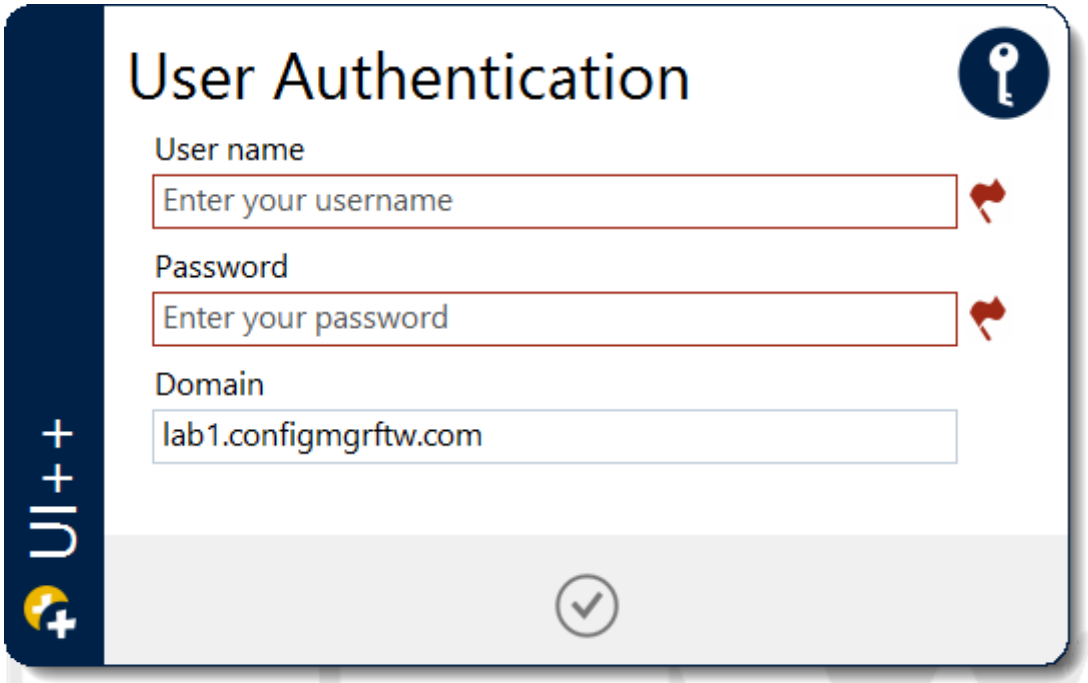


Figure 6: An AD Authentication Dialog

5.2 Registry

Values can be retrieved and put back into the registry. There is no UI for this task and there is no limit to the number of values that you can work with. Values retrieved are stored as strings.

Values written to the registry can be written as either string (REG_SZ) or number (REG_DWORD) values. If a key does not exist, it can be automatically created.

5.3 WMI

Values can be retrieved from WMI or written to WMI. Note that most of the default Windows WMI classes have very few writable attributes so writing information to WMI is of limited value except for custom data collection.

There is no UI for this task and there is no limit to the number of values that you can work with. Values retrieved are stored as strings.

5.4 Files

The first line of can be retrieved from a specified text file and optionally deleted. The value of this line is placed into a task sequence variable for use elsewhere within UI++ or a calling task sequence if used during OSD.

The scenario that this is designed for is providing a text file with a list of possible system names. This enables UI++ to provide a unique name to each new system when populating the OSDComputerName task sequence variable. There are other possible scenarios though.

To supply the text file to UI++ during OSD, first place the text file in a share accessible to systems that will execute UI++. Then use a Map Network Drive task before executing UI++ to map to this share using credentials that have read and write access to the text file. In the UI++ configuration file, specify the path to the text file using the drive letter specified in the Map Network Drive task as well as the proper sub-folders if necessary.

5.5 External Command

As its name implies, this action enables you to run any command-line that you wish. Nothing is explicitly returned to UI++ but running the command in the process of running UI++ enables UI++ to pass values to the command-line and then also use any task sequence variables that the command executed sets.

6 Running UI++

By default, UI++ loads its configuration from a file named *UI++.xml* using the normal file location rules. To load an alternately named file (or a file in an alternate location), use the command-line parameter discussed below. See section 9 (“Configuration File") for details.

6.1 Common Usage Scenarios

6.1.1 Use outside of a task sequence (including testing).

Simply run UI++ by double-clicking on the icon or from the command-line. The command-line parameters specified below are valid.

As mentioned above, task sequence variables are not used outside of a task sequence but there is no actual UI++ functionality loss – the internal variable system is equivalent and provides the same functionality in a transparent manner.

6.1.2 Use within a task sequence

Using UI++ within a task sequence is one of the main ways to add interactivity to a task sequence. The main drawback with this approach is that it may not the first thing that the users see. Also keep in mind that if you run a task sequence from within Windows, you need to inject processes that display UI, like UI++, onto the interactive desktop using something like ServuceUI.exe from MDT/UDI.

1. Make the UI++ files available to the task sequence. This includes *UI++.exe* and your configuration file -- the default name for the configuration file is *UI++.xml* but this can be changed using a command-line parameter (see Optional command-line parameters below).

Note: UI++ no longer requires oledlg.dll. This requirement was removed at some point in a recent build.

There are multiple ways to do this but I prefer to put them into a software distribution package – no program is necessary. This has the advantage in larger distributed environments of ensuring that the client accesses the files from the closest DP. The disadvantage is that you have to remember to update the DP anytime you update the configuration file. An alternative is to place the files in a shared folder and use a map folder task to make them available. This has the

advantage of not having to update the DP every time you make a change. It also has an advantage when your task sequence is set to download and execute because the command-line task to run UI++ (see the next step) can be run before the partition and format task within WinPE. A hybrid approach is also possible where you put UI++.exe in a package and make the configuration file available using a shared folder.

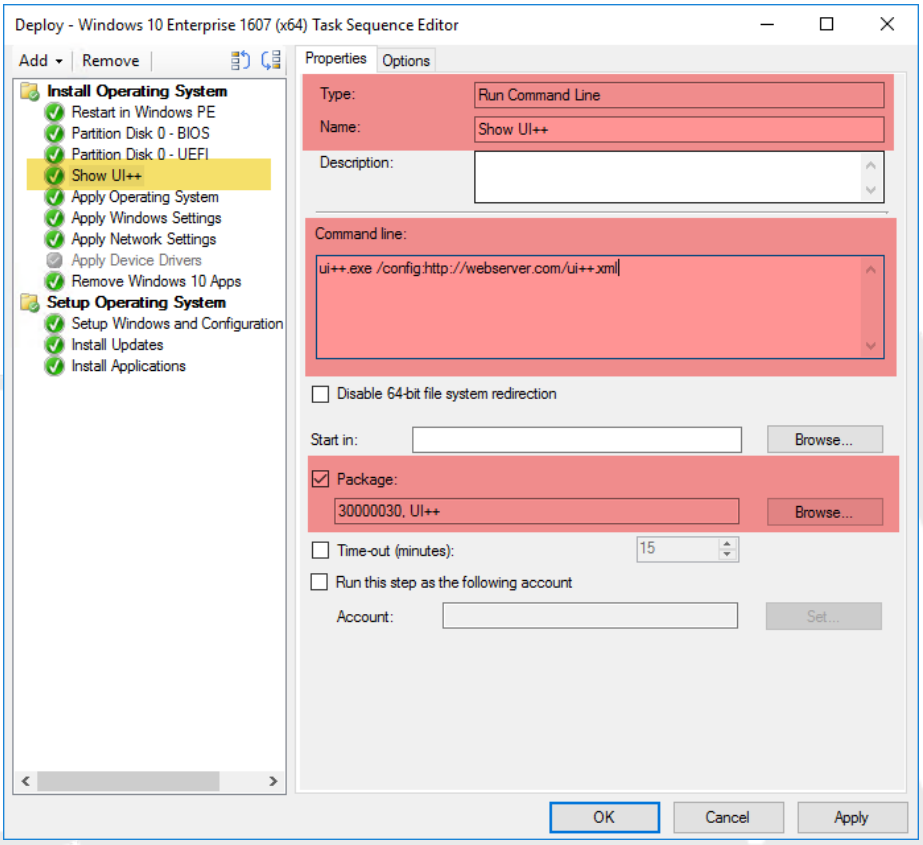


Figure 7: Using UI++ in a Task Sequence

2. Create a command-line task to run *UI++.exe*. If you used a package for step 1, reference the package you placed the files in this task. If you used a shared folder, make sure you used a map folder task before this one and prefix *UI++.exe* with the drive letter you mapped in that task.

6.1.3 Use as a prestart command

Instead of including UI++ in a task sequence explicitly, you can embed it in your boot image(s) and run it as a pre-start command. This has the advantage of making UI++ the first thing the user sees and also gives you the ability to set the *SMSTSPreferredAdvertID* task sequence variable using UI++ forcing a specific task sequence to be run.

As of ConfigMgr Current Branch, configuring a prestart for a boot image command is easy and is done completely in the console.

1. Create a UNC accessible folder with the necessary UI++ files in it.

2. Open the properties of the boot image that you wish to add a pre-start command to and go to the *Customization* tab.
    a. Check the *Enable prestart command* option.
    b. Supply a command-line to run UI++.
    c. Check the *Include files for the prestart command* option.
    d. Specify the UNC path to the folder with the UI++ files in it.
    e. Update the boot image.

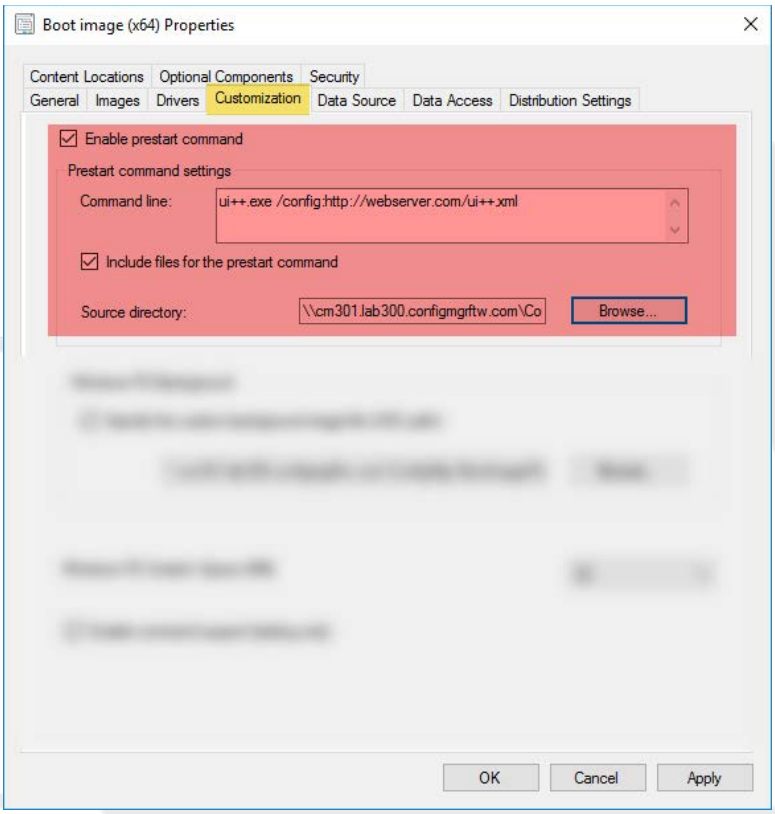


Figure 8: Configuring a boot image prestart command

You can also configure a prestart command for boot media during the *Create Task Sequence Media Wizard*. On the customization tab, simply do the following:

1. Check the *Enable prestart command* option.
2. Supply a command-line to run UI++.
3. Check the *Include files for the prestart command* option.
4. Choose a package containing the UI++ files. This package need not have a program, it is simply used to the copy the files from it.
5. Choose a distribution point where the package has been successfully distributed.

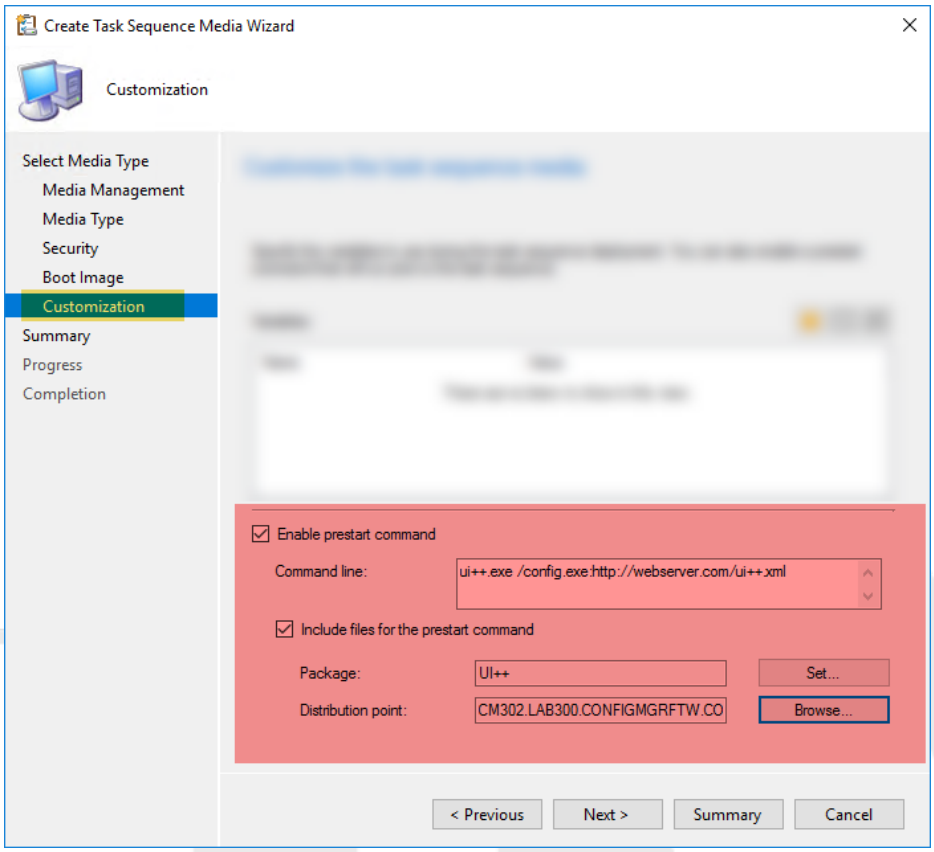


Figure 9: Configuring a prestart command in boot media

6.2 Optional command-line parameters

- **/config:<filename>** The filename of the configuration file to use. This can include a path if necessary. If not specified, *UI++.xml* will be used. In addition to loading a local configuration file, one can also be loaded from an HTTP or HTTPS location. To do this, specify the entire correct URL (including prefixing it with http:// or https://) where the xml file can be accessed from. UI++ will download the specified file from the URL to the current user's TEMP directory and then load the file from there.

  An advantage of using command-line arguments is that you can use other task sequence variables to set their values. For example, if you have multiple configuration files, you can populate a task sequence variable named MyConfig with the filename and then use the following command-line: **UI++.exe /config:%MyConfig%**.

  Another advantage to using this parameter is that you don't have to embed your configuration file in a package or boot media.

- **/retry:<count>** If you specify an http or https location for the configuration file using the **/config** switch, then this switch specifies how many times UI++ retries downloading the configuration file in the event of a file download failure. UI++ pauses for five seconds between each attempt.
- **/fallback:<filename>** If you specify an http or https location for the configuration file using the **/config** switch and the download fails, UI++ loads the file specified by this switch.

6.3 VARIABLE EDITOR

The Variable Editor is all new for UI++ 2.0 and enables the interactive user to view all set variables and modify them (except of course read-only variables which have been hardened from modification in ConfigMgr 2012). To open the variable editor, push Ctrl+F2 on any dialog. There are two lists presented, one for "Read-only" variables (shown in Figure 10) and one for "Editable" variables (shown in Figure 11).

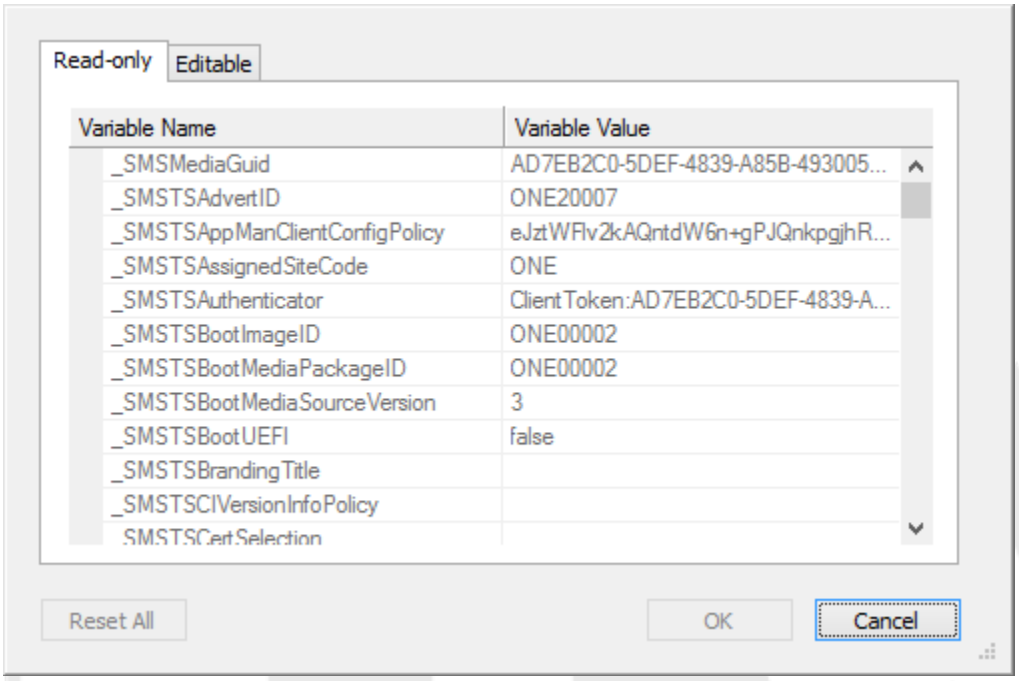


Figure 10: Read-only Variables

To modify an editable variable, simply click in the appropriate text box and modify the value. Note that your mileage may vary in modifying most of the built in variables. Each has a specific purpose and changing its value may result in various behaviors – this is not UI++'s fault. Know what the value is and does before changing it. In general, changing values during a task sequence from the variable editor has limited (if any) production value but can be very useful for testing purposes.

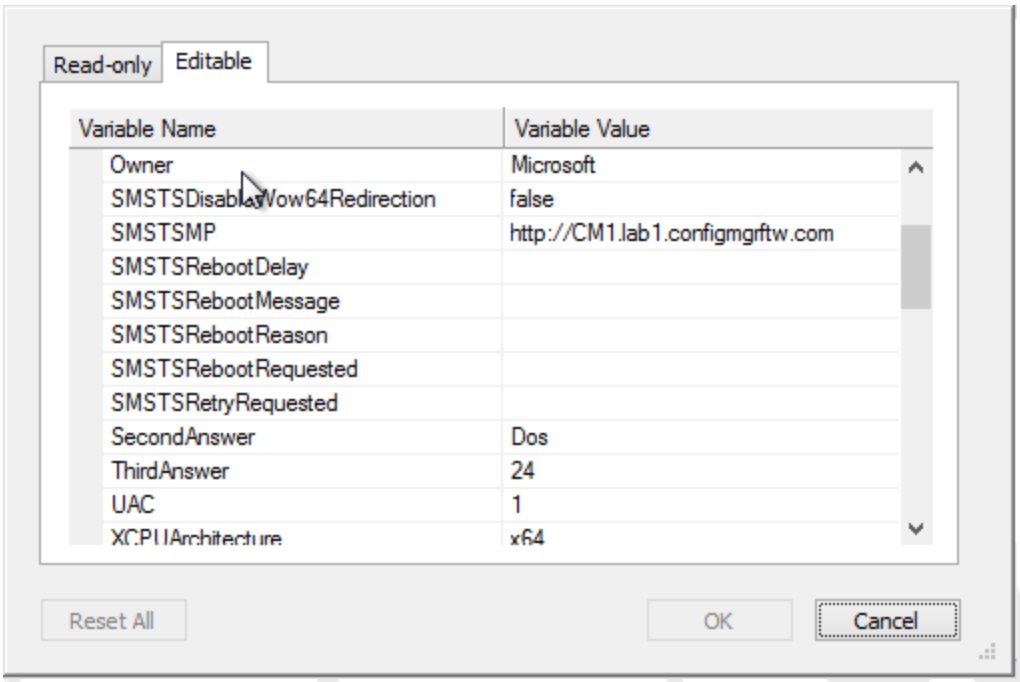


Figure 11: Editable Variables

To dump a complete list of variables and their values from with UI++, simply press Ctrl+F3 on any dialog. This will create a text file named **UI++ Variable Dump <Date> <Time>.txt** in the same place that the UI++ log file is currently being written to.

To completely disable the use of the variable editor, start UI++ using the **/disabletsvareditor** switch on the command-line. This will also disable dumping variables to a file.Log File

6.4 Log File

UI++ will generate or append to a log file named *UI++.log* every time it runs. This log records major events and all significant activity that UI++ performs. If you launch UI++ and nothing happens, check this log file for details. To prevent any ugliness from being presented to an end user, everything is sent to this file without notifying the interactive user.

The log file is a standard ConfigMgr log file and is best viewed using CMTrace. If UI++ is run inside a task sequence, the log file is located with the standard task sequence log file SMSTS.log: if you place the task at or near the beginning of the task sequence as described in the usage section, this will be *X:\Windows\Temp\SMSTS*. If you run UI++ outside of a task sequence, the log file will be located in the current user's temp directory which can easily be located using the environment variable *%TEMP%*.

7 Common Snippets

This section provides snippets of configuration XML for UI++ to address common scenarios or challenges.

7.1 Preflight Checks

```
<Action Type="DefaultValues" />
<Action Type="Preflight" Title="Preflight checks">
    <Check Text="WLAN Disconnected" CheckCondition='"%XWLANDisconnected%" = "True"' />
    <Check Text="Not on battery" CheckCondition='"%XOnBattery%" = "False"' />
    <Check Text="Minimum memory > 1GB" CheckCondition='%XHWMemory% >= 1024' />
    <Check Text="CPU Supports Windows 8+" CheckCondition='%XCPUPAE% AND %XCPUNX% AND
%XCPUSSE2% = True' />
</Action>
```

7.2 Required Applications Based on Group Membership

```
<Software>
   …
</Software>
<Action Type="UserAuth" Title="User Authentication" Domain="lab1.configmgrftw.com"
GetGroups="True" MaxRetryCount="5"/>
<Action Type="AppTree" Size="Tall" Title="Please choose your software">
  <SoftwareSets>
    <Set Name="Default">
      …
    </Set>
    <Set Name="Required" Condition='InStr ("%XAuthenticatedUserGroups%" &amp;amp; ",",
"Human Resources,") > 0'>
      <SoftwareGroup Id="abc123" Label="Human Resources Required Software"
Required="True">
        …
      </SoftwareGroup>
    </Set>
  </SoftwareSets>
</Action>
```

8 Complete Examples

The following examples present complete solutions using UI++. Note that these are just examples; the end-result can be achieved in multiple ways depending upon your goals. As with all things ConfigMgr, if what you come up with works, is valid (and supported), and meets your goals, then use it.

8.1 Example 1: Within OSD

This first example is for naming a system during OSD as well as specifying an OU and timezone based on a user selected location. The name for the system is the location code + W + the Dell Service tag. If no service tag is found, the interactive user is prompted to enter an equivalent value. The following three dialog boxes are shown – note that the second is only shown if a Dell service tag is not discovered:

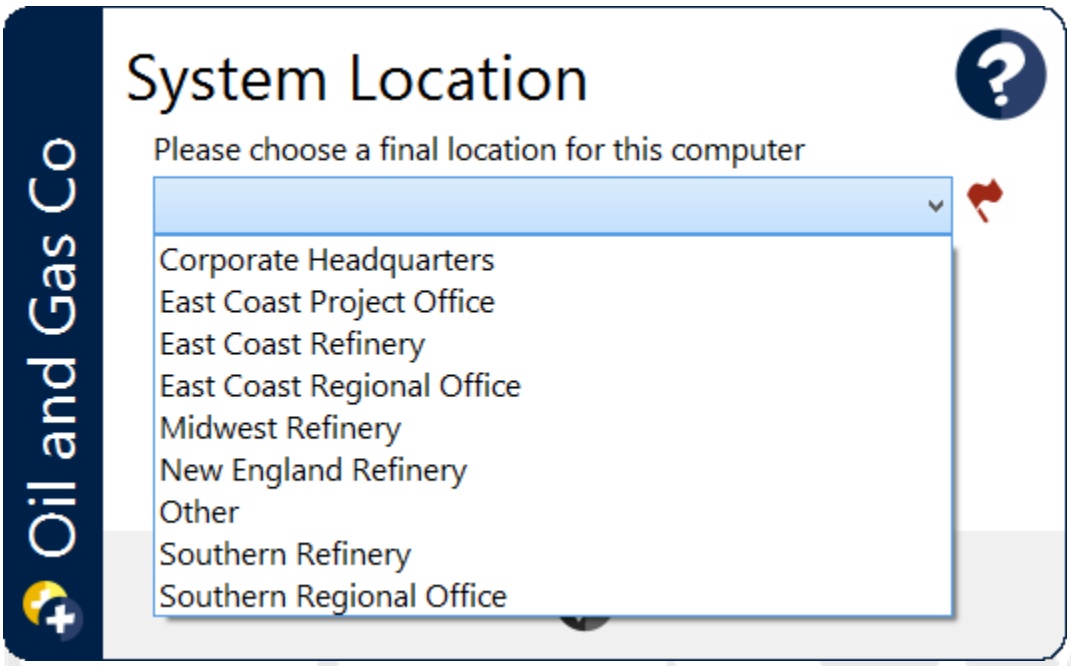


Figure 12: Example 1, Input Dialog for Location Entry

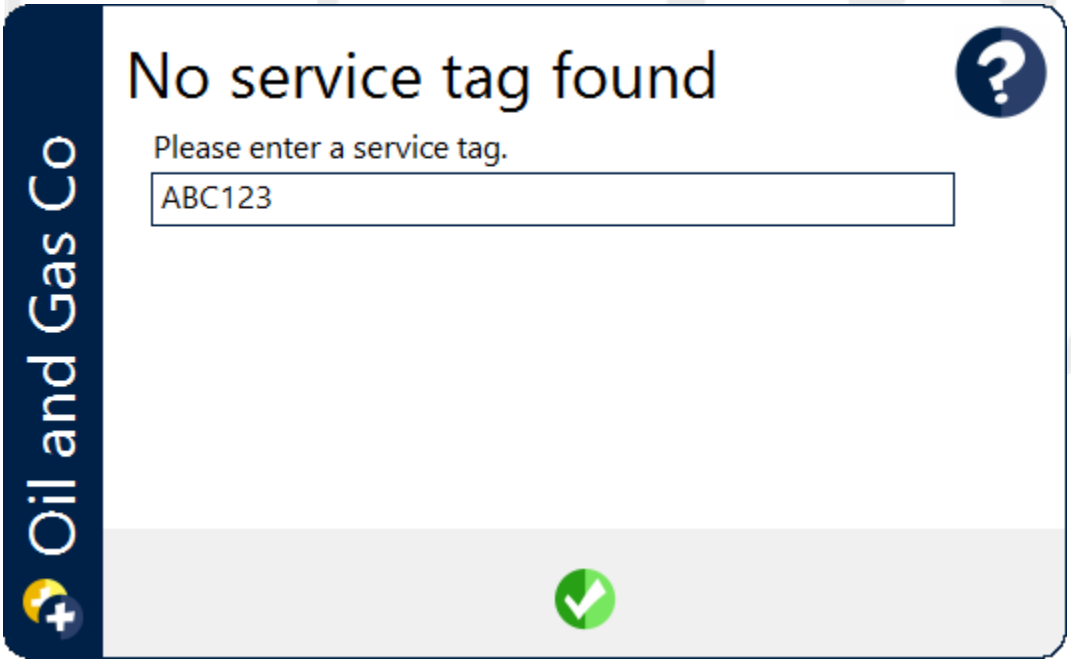


Figure 13: Example 1, Input Dialog for Service Tag Entry

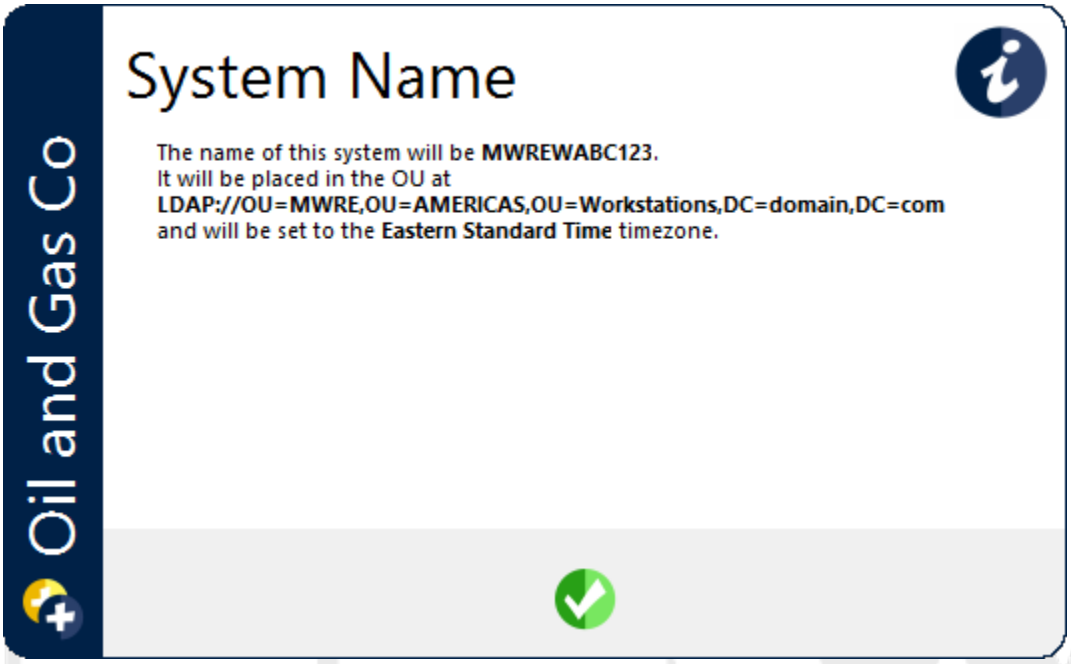


Figure 14: Example 1, Info Dialog for Final Information

8.1.1 XML Listing

The above dialogs are created using the following configuration XML:

```
<?xml version="1.0" encoding="utf-8"?>
<UIpp Title="Oil and Gas Co" Icon="UI++.ico">
  <Actions>
    <Action Type="DefaultValues" />
    <Action      Type="WMIRead"      Variable="ComputerName"      Namespace="root\cimv2"
Class="Win32_ComputerSystem" Property="Name"/>
    <Action      Type="WMIRead"      Variable="DellServiceTag"      Namespace="root\cimv2"
Class="Win32_SystemEnclosure" Property="SerialNumber"/>
    <Action Type="Input" Name="LocationChoice" Title="System Location">
      <ChoiceInput Variable="MyLocation" Question="Please choose a final location for this
computer" Required="True" >
        <Choice Option="East Coast Refinery" Value="ECRE"/>
        <Choice Option="East Coast Regional Office" Value="ECRO"/>
        <Choice Option="East Coast Project Office" Value="ECPO"/>
        <Choice Option="Midwest Refinery" Value="MWRE" />
        <Choice Option="Southern Refinery" Value="SORE" />
        <Choice Option="Southern Regional Office" Value="SORO" />
        <Choice Option="Corporate Headquarters" Value="COHQ" />
        <Choice Option="New England Refinery" Value="NERE"/>
        <Choice Option="Other" Value="MISC" />
      </ChoiceInput>
    </Action>
    <Action    Type="Input"    Name="ServiceTag"    Title="No    service    tag    found"
Condition='"%DellServiceTag%" = "" Or Len("%DellServiceTag%")'
      < 5 Or Len("%DellServiceTag%") > 7'>
      <TextInput Prompt="Service Tag" Hint="No service tag was found for this system,
please, enter one between 5 and 7 characters." Question="Please enter a service tag."
RegEx=".{5,7}" Variable="DellServiceTag" />
    </Action>
```

```
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "ECRE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "ECRO"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "ECPO"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "MWRE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "SORE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "SORO"'>Central
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "COHQ"'>Central
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "NERE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "MISC"'>Central
Standard Time</Action>
    <Action Type="TSVar" Name="OSDDomainOUName"
>LDAP://OU=%MyLocation%,OU=AMERICAS,OU=Workstations,DC=domain,DC=com</Action>
    <Action Type="TSVar" Name="OSDDomainOUName" Condition='"%MyLocation%" =
"MISC"'>LDAP://OU=Build,OU=Workstations,DC=domain,DC=com</Action>
    <Action Type="Info" Title="System Name" Name="SystemName">
      <![CDATA[The name of this system will be <b>%MyLocation%W%DellServiceTag%</b>.
It will be placed in the OU at <b>%OSDDomainOUName%</b>
and will be set to the <b>%OSDTimezone%</b> timezone.]]>
    </Action>
    <Action Type="TSVar" Name="OSDComputerName">%MyLocation%W%DellServiceTag%</Action>
  </Actions>
</UIpp>
```

8.1.2 XML Breakdown

To see exactly what's going on, the following listing breaks the XML down section by section with a detailed explanation.

1. `<?xml version="1.0" encoding="utf-8"?>`
   This is the default XML declaration specifying the version and the encoding of the file. Unless you have a reason to change it, also include this as is.

2. `<UIpp Title="Oil and Gas Co" Icon="UI++.ico">`
   This is the opening tag that begins defining functionality; this specifies the name to display in the sidebar, "Oil and Gas Co" in this example and an icon to use in the sidebar also. This name and icon appear in the sidebar for every dialog.

3. `<Actions>`
   The opening tag for all Actions.

4. `<Action Type="DefaultValues" />`
   The first action. This one gathers default values from the local system as defined in section 9.1.4 ("Default Values").

5. `<Action Type="WMIRead" Variable="ComputerName" Namespace="root\cimv2" Class="Win32_ComputerSystem" Property="Name"/>`
   `<Action Type="WMIRead" Variable="DellServiceTag" Namespace="root\cimv2" Class="Win32_SystemEnclosure" Property="SerialNumber"/>`

These two actions read a couple of values from WMI, specifically the current **Name** attribute from **Win32_ComputerSystem** WMI class and the **SerialNumber** attribute from the **Win32_SystemEnclosure** WMI class; both are located in the **root\cimv2** namespace. The **Name** attribute contains the current computer name of the system and the **SerialNumber** attribute is used by Dell to store the Dell service tag. These two value are stored in the variables named **ComputerName** and **DellServiceTag**, respectively, by UI++ for later use. If run from within OSD, the variables are task sequence variables.

6.
```
<Action Type="Input" Name="LocationChoice" Title="System Location">
  <ChoiceInput Variable="MyLocation" Question="Please choose a final location for this computer" Required="True" >
    <Choice Option="East Coast Refinery" Value="ECRE"/>
    <Choice Option="East Coast Regional Office" Value="ECRO"/>
    <Choice Option="East Coast Project Office" Value="ECPO"/>
    <Choice Option="Midwest Refinery" Value="MWRE" />
    <Choice Option="Southern Refinery" Value="SORE" />
    <Choice Option="Southern Regional Office" Value="SORO" />
    <Choice Option="Corporate Headquarters" Value="COHQ" />
    <Choice Option="New England Refinery" Value="NERE"/>
    <Choice Option="Other" Value="MISC" />
  </ChoiceInput>
</Action>
```

This action display an **Input** dialog as shown in Figure 12.(**Input** dialogs are discussed in more detail in section 9.1.8 (“Input”).) This particular dialog shows a single drop-box box, called a **ChoiceInput** in UI++. The text “Please choose a final location for this computer” is shown above the drop-box to prompt the user to make a choice. This drop-down contains choices for the nine different locations defined using the Choice elements and is required; i.e., the user must make a choice before being allowed to proceed. The value of whichever choice is selected by the user is stored in the variable name **MyLocation**.

7.
```
<Action Type="Input" Name="ServiceTag" Title="No service tag found" Condition='"%DellServiceTag%" = "" Or Len("%DellServiceTag%")' < 5 Or Len("%DellServiceTag%") > 7'>
  <TextInput Prompt="Service Tag" Hint="No service tag was found for this system, please, enter one between 5 and 7 characters." Question="Please enter a service tag." RegEx=".{5,7}" Variable="DellServiceTag" />
</Action>
```

This action shows the dialog from Figure 13. It shows a single text box and prompts the user to enter a service tag. Note the condition attribute on the Action element which dictates whether or not this action is processed at all. The condition checks for the existence of a value in the **DellServiceTag** variable (which was populated in step 5 above) or whether the string value in this variable is less than 5 characters long or more than 7 characters long. If any of these are true, then the action is skipped.

Also note the **RegEx** attribute on the **TextInput** element. This defines a regular expression; the value entered in the text box must match this regular expression in order for the user to proceed. In this example, the regular expression checks for a strength of length 5, 6, or 7.

8.
```
<Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "ECRE"'>Eastern Standard Time</Action>
<Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "ECRO"'>Eastern Standard Time</Action>
```

```
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "ECPO"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "MWRE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "SORE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "SORO"'>Central
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "COHQ"'>Central
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "NERE"'>Eastern
Standard Time</Action>
    <Action Type="TSVar" Name="OSDTimezone" Condition='"%MyLocation%" = "MISC"'>Central
Standard Time</Action>
```

This series of nearly identical actions uses conditions to check the value of the **MyLocation** variable set in step 6 when the user chose a location from the drop-down box. Only one of these Actions will ever be processed because **MyLocation** can only equal one of the values. The action that is processed sets the variable **OSDTimezone** to the proper value. Within a task sequence, the **OSDTimeZone** task sequence variable is used to directly populate the timezone value in unattend.xml and thus this will set the timezone used.

Note that the task of setting this variable could also have been done using alternate variables and alternate values in step 6 but this example show an alternate method that can be used for something more in-depth.

9.
```
<Action Type="TSVar" Name="OSDDomainOUName"
>LDAP://OU=%MyLocation%,OU=AMERICAS,OU=Workstations,DC=domain,DC=com</Action>
    <Action Type="TSVar" Name="OSDDomainOUName" Condition='"%MyLocation%" =
"MISC"'>LDAP://OU=Build,OU=Workstations,DC=domain,DC=com</Action>
```

These two actions set the value of the **OSDDomainOUName** variable (which similar to **OSDTimeZone** is used to populate the join OU in the unattend.xml file by OSD). The first action populates this variable using a standard LDAP path based upon the value of the **MyLocation** variable previously set. This first action is always run, however, in the case where the **MyLocation** value is "MISC", the second action will run and populate a completely different value for the OU (effectively over-writing what the first action set).

10.
```
<Action Type="Info" Title="System Name" Name="SystemName">
  <![CDATA[The name of this system will be <b>%MyLocation%W%DellServiceTag%</b>.
It will be placed in the OU at <b>%OSDOmainOUName%</b>
and will be set to the <b>%OSDTimezone%</b> timezone.]]>
    </Action>
```

This action display the final notification dialog shown in Figure 14.

11.
```
<Action Type="TSVar" Name="OSDComputerName">%MyLocation%W%DellServiceTag%</Action>
```

This final action sets the **OSDComputerName** variable which is used by the task sequence to set the actual computer name of the system being deployed.

12.
```
</Actions>
</UIpp>
```

These are the final closing tags: all elements in XML must have a closing tag or be closed.

8.2 Example 2: A Complex Real-world Example Within OSD

This is a real-world example with a mixture of different options and inputs, conditional options, as well as setting various standard task sequence variables to control what happens during the task sequence.

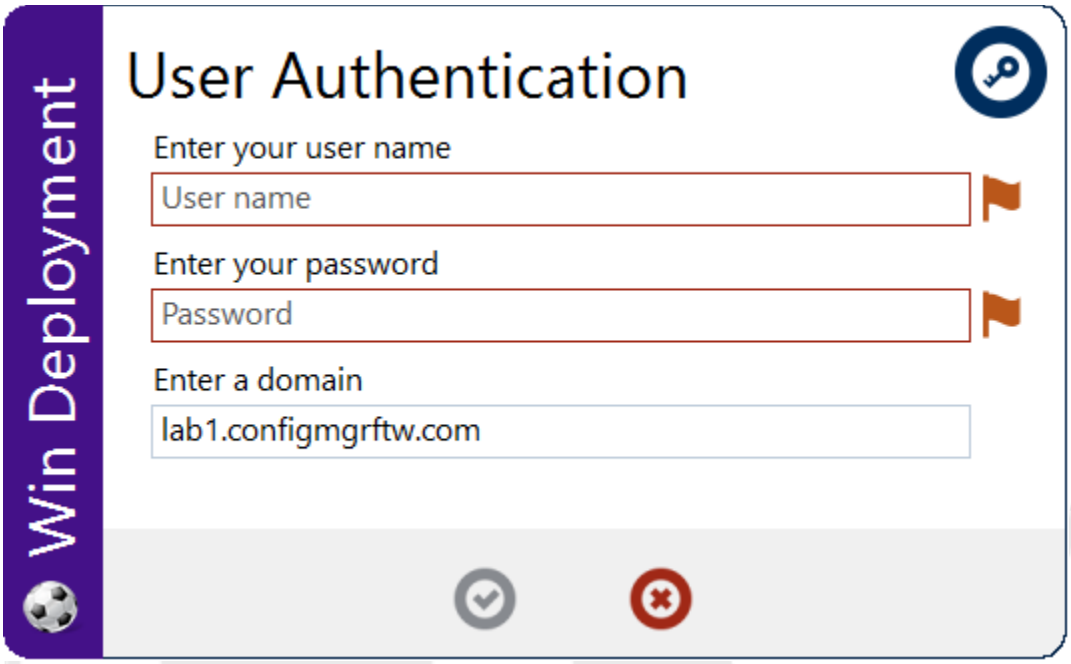


Figure 15: Example 2, User Authentication

Figure 16: Example 2, A simple welcome dialog box showing the authenticated user's name and a banner image

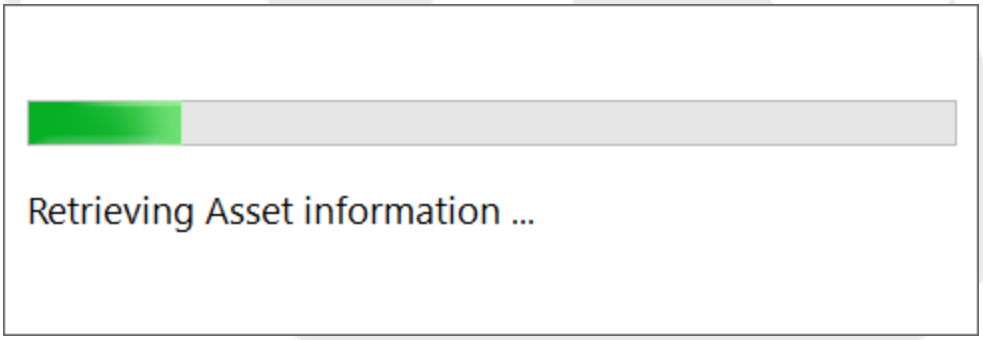


Figure 17: Example 2, Progress bar during the DefaultValues action

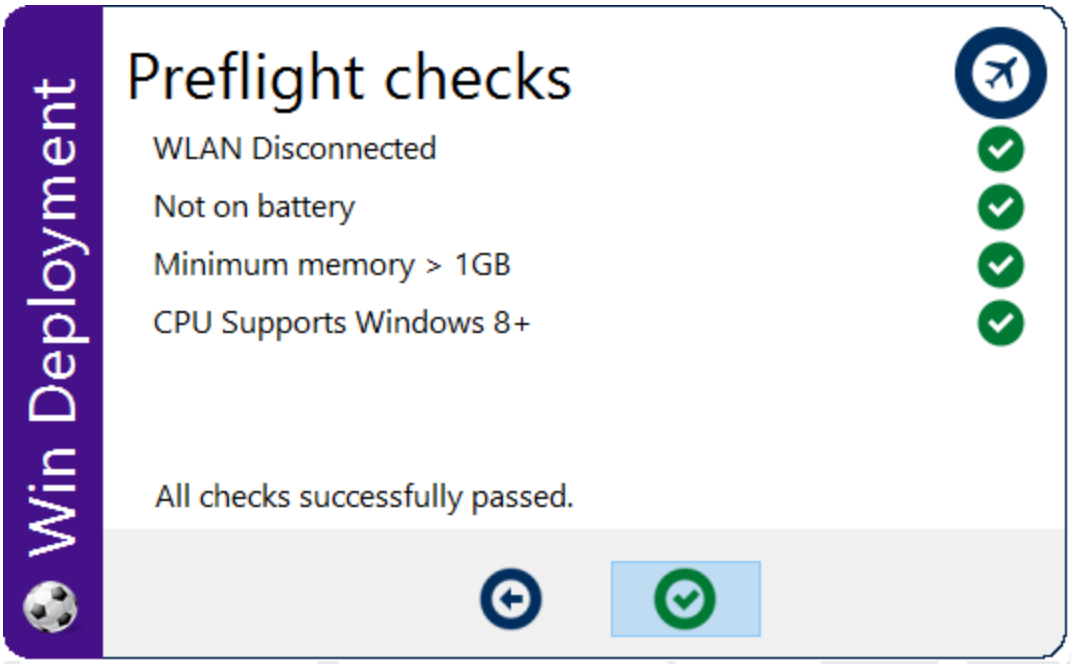


Figure 18: Example 2, Preflight checks all passed

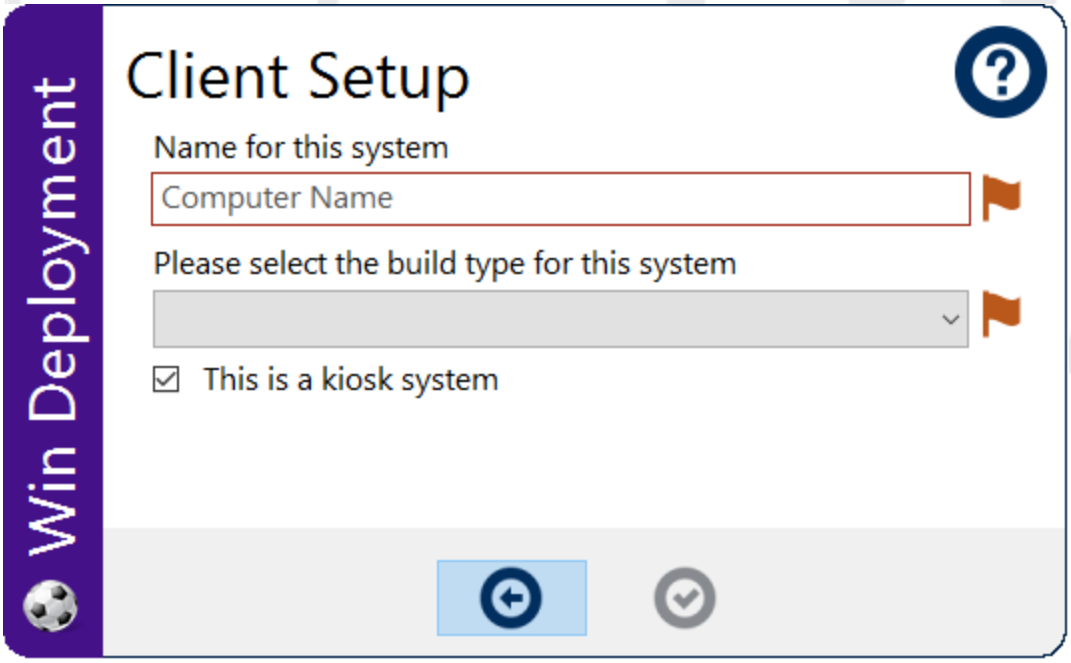


Figure 19: Example 2, A simple data collection dialog showing all three possible types of user input

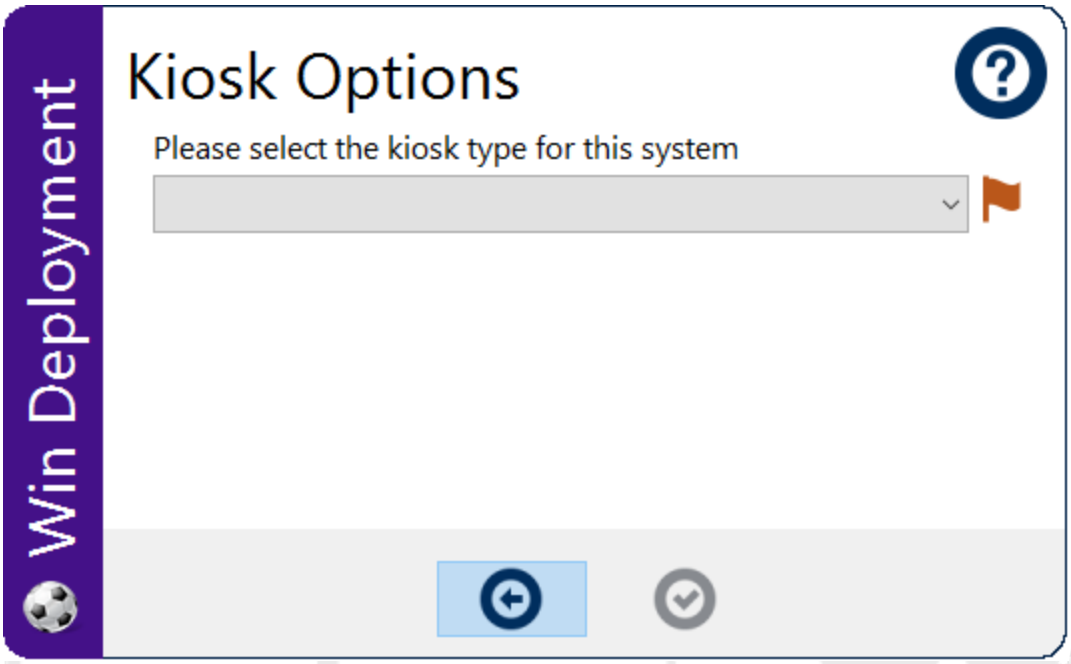


Figure 20: Example 2, KIOSK Option Dialog

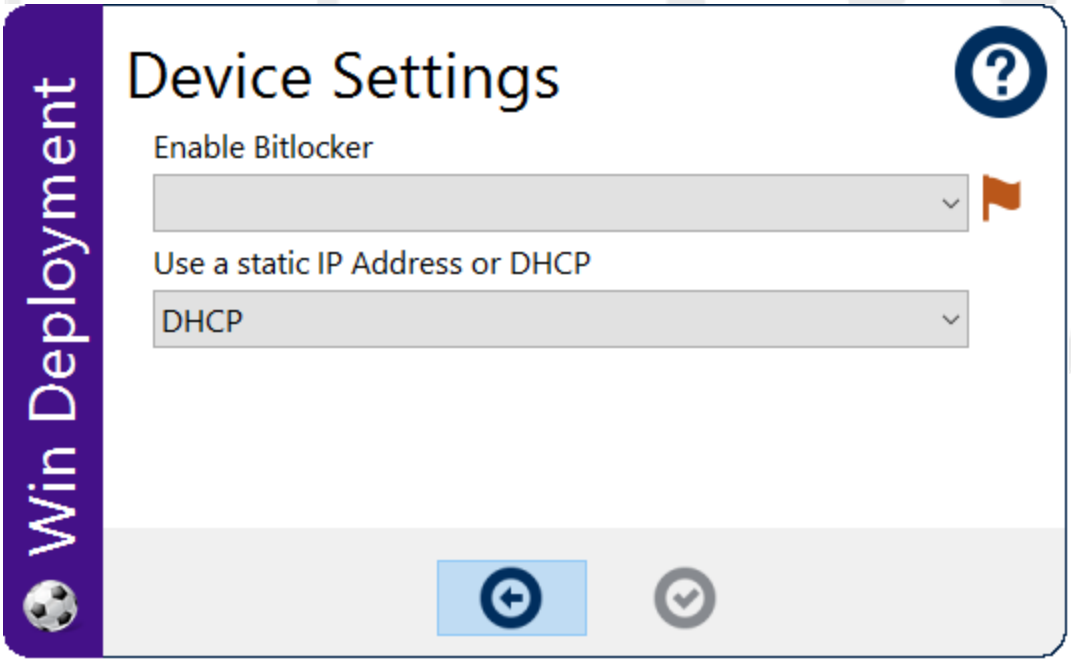


Figure 21: Example 2, Device Settings for a kiosk system

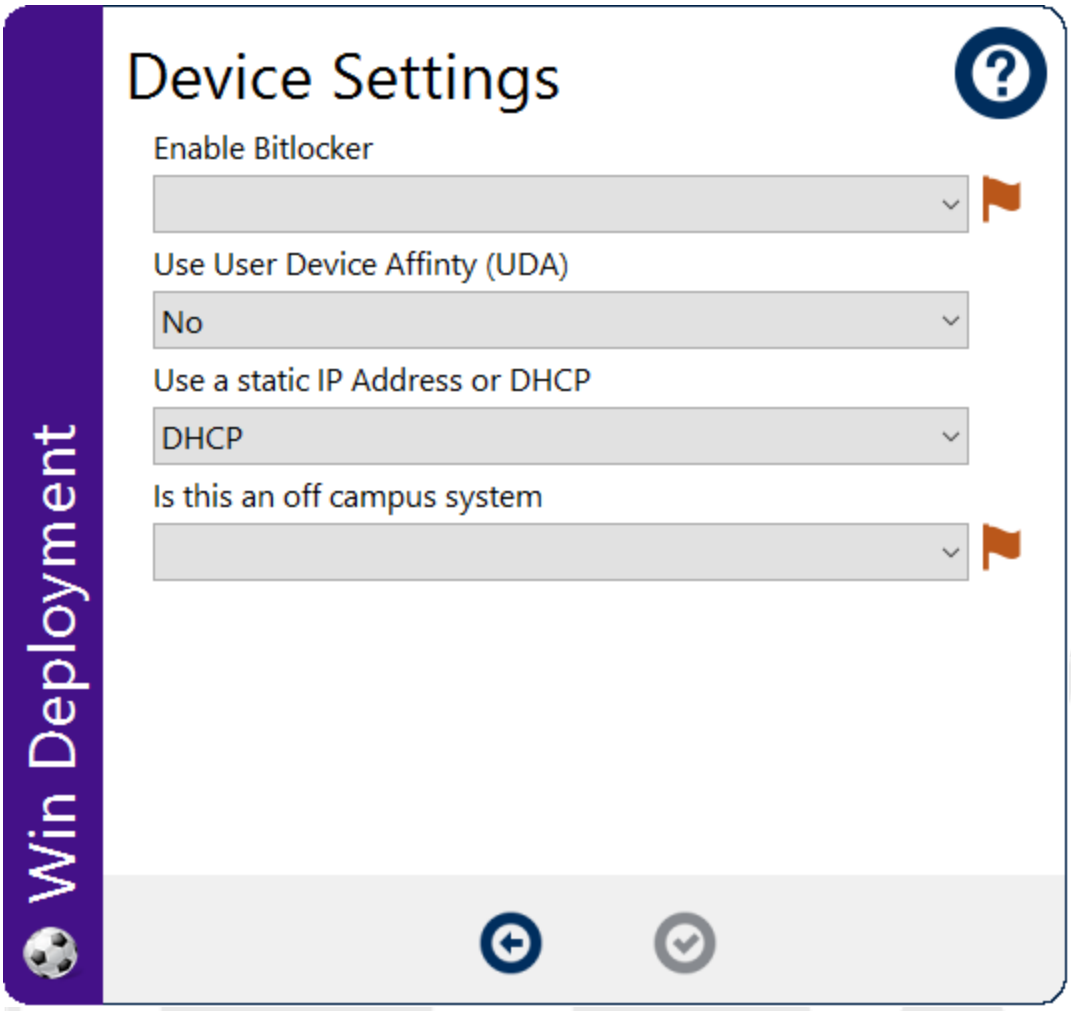


Figure 22: Example 2, Device Settings for a non-kiosk, non-developer system

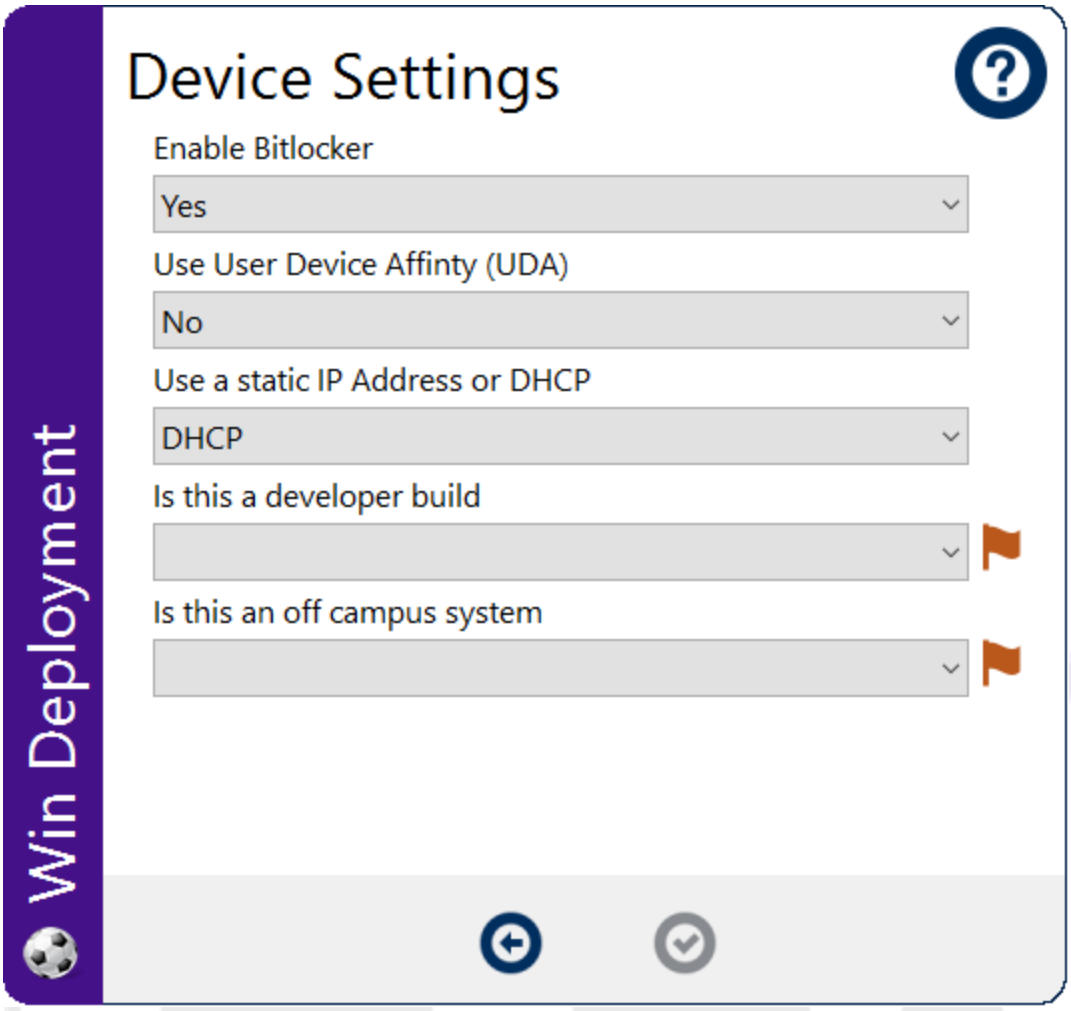


Figure 23:Example 2, Device Settings for a non-kiosk, developer possible system

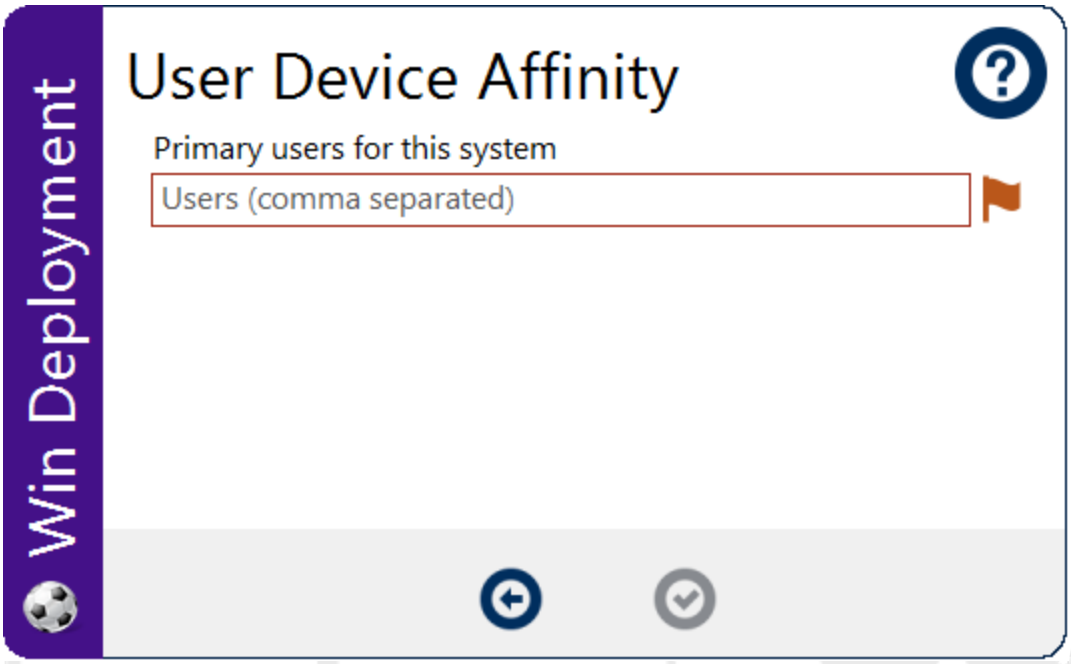


Figure 24: Example 2, User Device Affinity input

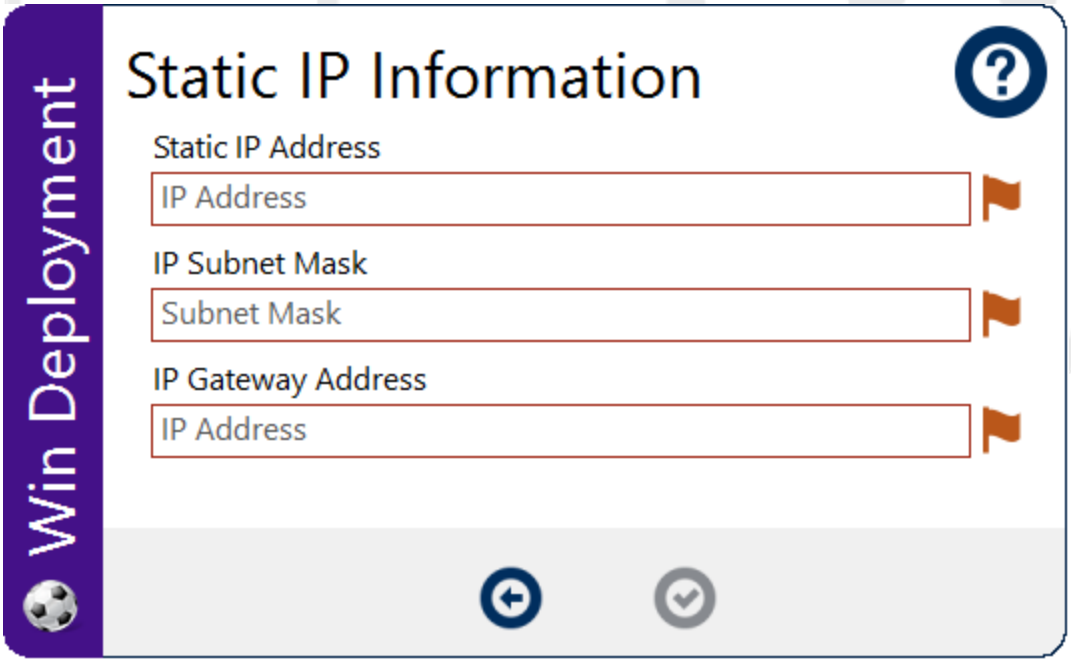


Figure 25: Example 2, Static IP Address Input

8.2.1 XML Listing

```
<?xml version="1.0" encoding="utf-8"?>
<UIpp Title="Win Deployment" Icon="my.ico" Color="#441188" DialogIcons="Yes">
        <Actions>
    <Action Type="UserAuth" Title="User Authentication"
Domain="lab1.configmgrftw.com" GetGroups="False" MaxRetryCount="5" />
    <Action Type="Info" Name="myInfo" Title="Welcome" Image="my.png"
ShowCancel="True">
```

```
      <![CDATA[Welcome %XAuthenticatedUser%]]>
    </Action>
    <Action Type="DefaultValues" ValueTypes="All" ShowProgress="True">
      <Text Type="OS" Value="Retrieving Operating System information" />
      <Text Type="TPM" Value="Retrieving TPM information" />
      <Text Type="Net" Value="Retrieving Networking information" />
      <Text Type="VM" Value="Retrieving Virtualization information" />
      <Text Type="Asset" Value="Retrieving Asset information" />
      <Text Type="Domain" Value="Retrieving Domain information" />
    </Action>
    <Action Type="Preflight" Title="Preflight checks" ShowBack="True">
      <Check Text="WLAN Disconnected" CheckCondition='"%XWLANDisconnected%" =
"True"' />
      <Check Text="Not on battery" CheckCondition='"%XOnBattery%" = "False"' />
      <Check Text="Minimum memory > 1GB" CheckCondition='%XHWMemory% >= 1024' />
      <Check Text="CPU Supports Windows 8+" CheckCondition='%XCPUPAE% AND %XCPUNX%
AND %XCPUSSE2% = True' />
    </Action>

    <Action Type="Input" Name="ClientSetupInput" Title="Client Setup"
ShowBack="True">
      <TextInput Prompt="Computer Name" Hint="Enter the name for this system"
RegEx="[^\&amp;quot;/\\\[\]:;\|=,\+\*\?&gt;&lt;]{3,15}" Variable="ZZComputerName"
Question="Name for this system" />
      <ChoiceInput Variable="ZZBuildType" Question="Please select the build type
for this system" Required="True">
        <Choice Option="Windows 10 (x64)" Value="Win10x64" />
        <Choice Option="Windows 8.1 (x64)" Value="Win8.1x64" />
        <Choice Option="Windows 7 (x64)" Value="Win7x64" />
      </ChoiceInput>
      <CheckboxInput Variable="ZZKiosk" Question="This is a kiosk system"
CheckedValue="True" UncheckedValue="False" Default="True"/>
    </Action>

    <Action Type="TSVar" Name="ZZDeveloper"
Condition='LCase(Left("%ZZComputerName%", 4)) &lt;&gt; "dev-"'>"False"</Action>

    <Action Type="Input" Name="KioskOptionsInput" Title="Kiosk Options"
ShowBack="True" Condition='"%ZZKiosk%" = "True"'>
      <ChoiceInput Variable="ZZKioskType" Question="Please select the kiosk type
for this system" Required="True">
        <Choice Option="E-mail" Value="E-mail" />
        <Choice Option="Time" Value="Time" />
        <Choice Option="Other" Value="Other" />
      </ChoiceInput>
    </Action>

    <Action Type="TSVar" Name="ZZBitLocker" Condition='"%ZZKioskType%" =
"Other"'>"YES"</Action>

    <Action Type="TSVar" Name="UIppDeviceSettingsSize"
Condition='LCase(Left("%ZZComputerName%", 4)) = "dev-" Or "%ZZKiosk%" =
"False"'>"Tall"</Action>
    <Action Type="TSVar" Name="UIppDeviceSettingsSize"
Condition='LCase(Left("%ZZComputerName%", 4)) &lt;&gt; "dev-" And "%ZZKiosk%" =
"True"'>"Regular"</Action>
```

```
    <Action Type="Input" Size="%UIppDeviceSettingsSize%"
Name="DeviceSettingsInput" Title="Device Settings" ShowBack="True">
      <ChoiceInput Variable="ZZBitLocker" Question="Enable Bitlocker"
Required="True">
        <Choice Option="Yes" Value="YES" />
        <Choice Option="No" Value="NO" />
      </ChoiceInput>
      <ChoiceInput Variable="ZZUDA" Question="Use User Device Affinty (UDA)"
Required="True" Default="False" Condition='"%ZZKiosk%" = "False"'>
        <Choice Option="Yes" Value="True" />
        <Choice Option="No" Value="False" />
      </ChoiceInput>
      <ChoiceInput Variable="ZZDHCP" Question="Use a static IP Address or DHCP"
Required="True" Default="DHCP">
        <Choice Option="DHCP" Value="True" />
        <Choice Option="Static IP" Value="False" />
      </ChoiceInput>
      <ChoiceInput Variable="ZZDeveloper" Question="Is this a developer build"
Required="True" Condition='LCase(Left("%ZZComputerName%", 4)) = "dev-"'>
        <Choice Option="Yes" Value="True" />
        <Choice Option="No" Value="False" />
      </ChoiceInput>
      <ChoiceInput Variable="ZZHomeSystem" Question="Is this an off campus system"
Required="True" Condition='"%ZZKiosk%" = "False"'>
        <Choice Option="Yes" Value="True" />
        <Choice Option="No" Value="False" />
      </ChoiceInput>
    </Action>

    <Action Type="Input" Name="UDAInput" Title="User Device Affinity"
ShowBack="True" Condition='"%ZZUDA%" = "True"'>
      <TextInput Prompt="Users (comma separated)" Hint="Enter the primary users
for this system (comma separated)" Variable="ZZUDAUsers" Question="Primary users
for this system" />
    </Action>

    <Action Type="Input" Name="StaticIPInput" Title="Static IP Information"
ShowBack="True" Condition='"%ZZDHCP%" = "False"'>
      <TextInput Prompt="IP Address" Hint="Enter the static IP Address for this
system" Variable="ZZIPAddress" Question="Static IP Address" RegEx="((25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"/>
      <TextInput Prompt="Subnet Mask" Hint="Enter the subnet mask for this system"
Variable="ZZIPSubnet" Question="IP Subnet Mask" RegEx="((25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"/>
      <TextInput Prompt="IP Address" Hint="Enter the gateway for this system"
Variable="ZZIPGateway" Question="IP Gateway Address" RegEx="((25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"/>
    </Action>

    <Action Type="TSVar" Name="BDEInstallSuppress" >"%ZZBitLocker%"</Action>

    <Action Type="TSVar" Name="OSDAdapter0IPAddressList" Condition='"%ZZDHCP%" =
"False"'>"%ZZIPAddress%"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0SubnetMask" Condition='"%ZZDHCP%" =
"False"'>"%ZZIPSubnet%"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0Gateways" Condition='"%ZZDHCP%" =
"False"'>"%ZZIPGateway%"'</Action>
```

```
    <Action Type="TSVar" Name="OSDAdapter0EnableDHCP" Condition='"%ZZDHCP%" =
"False"'>"False"'</Action>
    <Action Type="TSVar" Name="OSDAdapterCount" Condition='"%ZZDHCP%" =
"False"'>"1"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0EnableWINS" Condition='"%ZZDHCP%" =
"False"'>"True"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0WINSServerList" Condition='"%ZZDHCP%" =
"False"'>"10.10.1.200,10.10.5.200"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0DNSServerList" Condition='"%ZZDHCP%" =
"False"'>"10.10.1.1,10.10.5.100,10.10.8.50"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0EnableDNSRegistration"
Condition='"%ZZDHCP%" = "False"'>"True"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0DNSDomain" Condition='"%ZZDHCP%" =
"False"'>"lab1.configmgrftw.com"'</Action>

    <Action Type="TSVar" Name="SMSTSUdaUsers" >"%ZZUDAUsers%"</Action>
    <Action Type="TSVar" Name="OSDComputerName" >"%ZZComputerName%"</Action>
    <Action Type="TSVar" Name="OSDBuildType" >"%ZZBuildType%"</Action>

    <Action Type="TSVar" Name="OSDType" Condition='"%ZZKiosk%" =
"True"'>"%ZZKioskType%"</Action>
    <Action Type="TSVar" Name="OSDType" Condition='"%ZZKiosk%" =
"False"'>"STD"</Action>
    <Action Type="TSVar" Name="OSDType" Condition='"%ZZHomeSystem%" =
"True"'>"Home"</Action>
    <Action Type="TSVar" Name="OSDType" Condition='"%ZZDeveloper%" =
"True"'>"DEV"</Action>

  </Actions>
</UIpp>
```

8.2.2 XML Breakdown

1. `<?xml version="1.0" encoding="utf-8"?>`

This is the default XML declaration specifying the version and the encoding of the file. Unless you have a reason to change it, also include this as is.

2. `<UIpp Title="Win Deployment" Icon="my.ico" Color="#441188" DialogIcons="Yes">`

This is the opening tag that begins defining functionality; this specifies the name to display in the sidebar, “Windows Deployment” in this example, an icon to use in the sidebar, and the color of the sidebar also. This name and icon appear in the sidebar for every dialog. Additional, decorative icons are displayed on every dialog corresponding to the functionality of the dialog/action.

3. `<Actions>`

The opening tag for all Actions.

4. `<Action Type="UserAuth" Title="User Authentication" Domain="lab1.configmgrftw.com" GetGroups="False" MaxRetryCount="5" />`

The first action. As shown in Figure 15, this action prompts the interactive user to authenticate against the lab1.configmgrftw.com active directory domain. The interactive user will only be allowed to try to authenticate 5 times and group memberships will not be collected.

5.
```
<Action Type="Info" Name="myInfo" Title="Welcome" Image="my.png"
ShowCancel="True">
  <![CDATA[Welcome %XAuthenticatedUser%]]>
</Action>
```

Shows a generic welcome dialog box as shown in Figure 16. The user name entered in the previous action used for AD authentication is used to populate the message in the dialog box and a cancel button is shown to allow the user to cancel UI++. If canceled, UI++ will return an error code of 1223 (0x4c7) which is a standard Windows error code meaning "The operation was canceled by the user." This in turn will cause the task in the task sequence to error which may cause the entire task sequence to fail.

```
6.  <Action Type="DefaultValues" ValueTypes="All" ShowProgress="True">
      <Text Type="OS" Value="Retrieving Operating System information" />
      <Text Type="TPM" Value="Retrieving TPM information" />
      <Text Type="Net" Value="Retrieving Networking information" />
      <Text Type="VM" Value="Retrieving Virtualization information" />
      <Text Type="Asset" Value="Retrieving Asset information" />
      <Text Type="Domain" Value="Retrieving Domain information" />
    </Action>
```

Gathers all default values from the local system as defined in section 9.1.4 ("Default Values"). Figure 17 shows the optional progress bar dialog to ensure the user is aware of some background processing taking place – customized messages are provided for each of the different types of data collected during this action for display in the dialog.

```
7.  <Action Type="Preflight" Title="Preflight checks" ShowBack="True">
      <Check Text="WLAN Disconnected" CheckCondition='"%XWLANDisconnected%" = "True"'
/>
      <Check Text="Not on battery" CheckCondition='"%XOnBattery%" = "False"' />
      <Check Text="Minimum memory > 1GB" CheckCondition='%XHWMemory% >= 1024' />
      <Check Text="CPU Supports Windows 8+" CheckCondition='%XCPUPAE% AND %XCPUNX% AND
%XCPUSSE2% = True' />
    </Action>
```

Performs 4 pre-flights checks based upon values collected by the **DefaultActions** action just prior to this one – without the **DefaultActions** task preceding this one, none of these checks would pass. The CheckCondition attributes define VBScript conditions that are evaluated to determine whether a check passes or not. If comparing strings, ensure that they are enclosed in double-quotes otherwise VBScript won't treat the values as strings.

The back-button is also enabled on this action. If the user presses the back button, the **Info** welcome dialog is re-shown, the **DefaultValues** action is not re-executed though until after the user presses OK on the **Info** action. Thus this does result in the default values being re-collected so that if anything has changed that would affect any of the checks, it would be properly reflected. For example, if the system was not plugged into AC power when the user got to this action, a red X would be shown for the second check. If the user plugs the system in, hits the back button, and then presses OK on the welcome dialog, the pre-flight action will now properly reflect that the system is plugged into AC power and allow the user to proceed. Because of this, it's a good idea to place a generic information dialog like this before the **DefaultValues** task otherwise there would be nowhere to go back to. You also wouldn't want to place a **UserAuth** action before the **DefaultValues** action as this would force the user to re-authenticate.

```
8.  <Action Type="Input" Name="ClientSetupInput" Title="Client Setup" ShowBack="True">
      <TextInput Prompt="Computer Name" Hint="Enter the name for this system"
RegEx="[^\&amp;quot;/\\\[\]:;\|=,\+\*\?&amp;gt;&amp;lt;]{3,15}" Variable="ZZComputerName"
Question="Name for this system" />
      <ChoiceInput Variable="ZZBuildType" Question="Please select the build type for
this system" Required="True">
```

```
        <Choice Option="Windows 10 (x64)" Value="Win10x64" />
        <Choice Option="Windows 8.1 (x64)" Value="Win8.1x64" />
        <Choice Option="Windows 7 (x64)" Value="Win7x64" />
      </ChoiceInput>
      <CheckboxInput Variable="ZZKiosk" Question="This is a kiosk system"
CheckedValue="True" UncheckedValue="False" Default="True"/>
    </Action>
```

As shown in Figure 19, this action displays a user data collection dialog with all three different types of input types used: a text input field, a drop list and a checkbox. The text input box only allows valid characters for NetBIOS names of length 3 to 15. The drop list shows three different options and the checkbox is used to dictate what additional user input items are shown in subsequent dialogs.

9. `<Action Type="TSVar" Name="ZZDeveloper" Condition='LCase(Left("%ZZComputerName%", 4)) <> "dev-"'>"False"</Action>`

This sets the ZZDeveloper task sequence variable to False if the computer name provided in the last action by the user is not prefixed with "dev-". This must be done for cases where the back button is used to get back to this dialog; without this it is possible that the variable is set to True in step 13 below but the user changes their mind, comes back to this dialog, and changes the system name away from this convention.

Note the use of the **LCase** and **Left** VBScript functions in the VBScript expression used for the condition to ensure the comparison is case insensitive and to only the first 4 characters of the string.

10.
```
<Action Type="Input" Name="KioskOptionsInput" Title="Kiosk Options"
    ShowBack="True" Condition='"%ZZKiosk%" = "True"'>
   <ChoiceInput Variable="ZZKioskType" Question="Please select the kiosk type for this
system" Required="True">
      <Choice Option="E-mail" Value="E-mail" />
      <Choice Option="Time" Value="Time" />
      <Choice Option="Other" Value="Other" />
   </ChoiceInput>
 </Action>
```

This simple dialog, shown in Figure 20, is only shown if the kiosk checkbox from the previous action is selected; this is determined by the VBScript comparison expression defined in the Condition attribute.

11. `<Action Type="TSVar" Name="ZZBitLocker" Condition='"%ZZKioskType%" = "Other"'>"YES"</Action>`

These actions set the Task Sequencer Variable ZZBitlocker to YES if the Kiosk type chosen in the previous action is "Other".

12.
```
<Action Type="TSVar" Name="UIppDeviceSettingsSize"
    Condition='LCase(Left("%ZZComputerName%", 4)) = "dev-" Or "%ZZKiosk%" =
    "False"'>"Tall"</Action>
        <Action Type="TSVar" Name="UIppDeviceSettingsSize"
    Condition='LCase(Left("%ZZComputerName%", 4)) &lt;&gt; "dev-" And "%ZZKiosk%" =
    "True"'>"Regular"</Action>
```

These two actions set the value of the UIppDeviceSettingSize task sequence variable. This variable is used in the next action to determine whether a tall or regular dialog will be shown. If the system is possible a developer system (determined by the presence of a "dev-" prefix of the computer name entered in step 8 or is not a kiosk (determined by the checkbox choice made in step 8, then the next action should show a tall dialog box to accommodate additional input. If the opposite is true, then the number of items is not as great and a regular dialog can be used.

Note the use of <> in the condition of the second action above. These are XML entities that equate to < and > (angle brackets or less than and greater than symbols). The actual symbols cannot be directly used because they have special meaning in XML and would throw off the XML parser.

13.
```
<Action Type="Input" Size="%UIppDeviceSettingsSize%" Name="DeviceSettingsInput"
   Title="Device Settings" ShowBack="True">
     <ChoiceInput Variable="ZZBitLocker" Question="Enable Bitlocker" Required="True">
       <Choice Option="Yes" Value="YES" />
       <Choice Option="No" Value="NO" />
     </ChoiceInput>
     <ChoiceInput Variable="ZZUDA" Question="Use User Device Affinty (UDA)"
Required="True" Default="False" Condition='"%ZZKiosk%" = "False"'>
       <Choice Option="Yes" Value="True" />
       <Choice Option="No" Value="False" />
     </ChoiceInput>
     <ChoiceInput Variable="ZZDHCP" Question="Use a static IP Address or DHCP"
Required="True" Default="DHCP">
       <Choice Option="DHCP" Value="True" />
       <Choice Option="Static IP" Value="False" />
     </ChoiceInput>
     <ChoiceInput Variable="ZZDeveloper" Question="Is this a developer build"
Required="True" Condition='LCase(Left("%ZZComputerName%", 4)) = "dev-"'>
       <Choice Option="Yes" Value="True" />
       <Choice Option="No" Value="False" />
     </ChoiceInput>
     <ChoiceInput Variable="ZZHomeSystem" Question="Is this an off campus system"
Required="True" Condition='"%ZZKiosk%" = "False"'>
       <Choice Option="Yes" Value="True" />
       <Choice Option="No" Value="False" />
     </ChoiceInput>
   </Action>
```

Based upon the options chosen in previous dialogs, this action shows a dialog with varying inputs in it; these are shown in Figure 21, Figure 22, and Figure 23. The different user inputs are added based upon the condition defined on each.

Note that drop-lists were used instead of checkboxes for the Yes/No questions. This was done before the checkbox capability was added to UI++ and is perfectly valid – which method you use is up to you and depends upon what you want to show to the user.

14.
```
<Action Type="Input" Name="UDAInput" Title="User Device Affinity" ShowBack="True"
   Condition='"%ZZUDA%" = "True"'>
     <TextInput Prompt="Users (comma separated)" Hint="Enter the primary users for
this system (comma separated)" Variable="ZZUDAUsers" Question="Primary users for this
system" />
   </Action>
```

This action displays a very simple dialog with one, free text input box. This dialog is only shown if the drop list to use user device affinity is set to Yes from step 13. This option is only available on non-kiosk systems.

15.
```
<Action Type="Input" Name="StaticIPInput" Title="Static IP Information"
   ShowBack="True" Condition='"%ZZDHCP%" = "False"'>
```

```
      <TextInput Prompt="IP Address" Hint="Enter the static IP Address for this
system" Variable="ZZIPAddress" Question="Static IP Address" RegEx="((25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"/>
      <TextInput Prompt="Subnet Mask" Hint="Enter the subnet mask for this system"
Variable="ZZIPSubnet" Question="IP Subnet Mask" RegEx="((25[0-5]|2[0-4][0-9]|[01]?[0-
9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"/>
      <TextInput Prompt="IP Address" Hint="Enter the gateway for this system"
Variable="ZZIPGateway" Question="IP Gateway Address" RegEx="((25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"/>
    </Action>
```

This action displays a dialog with three text input boxes for entering IP information as shown in Figure 25. Each text box has a regular expression ensuring the entered text is in the form of a proper IP address. This dialog is only shown if the drop list for choosing DHCP is set to use Static IP from step 13.

16. `<Action Type="TSVar" Name="BDEInstallSuppress" >"%ZZBitLocker%"</Action>`

This action sets a task sequence variable name BDEInstallSuppress that is used within the Task Sequence itself to enable or disable BitLocker.

17.
```
<Action Type="TSVar" Name="OSDAdapter0IPAddressList" Condition='"%ZZDHCP%" =
    "False"'>"%ZZIPAddress%"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0SubnetMask" Condition='"%ZZDHCP%" =
"False"'>"%ZZIPSubnet%"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0Gateways" Condition='"%ZZDHCP%" =
"False"'>"%ZZIPGateway%"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0EnableDHCP" Condition='"%ZZDHCP%" =
"False"'>"False"'</Action>
    <Action Type="TSVar" Name="OSDAdapterCount" Condition='"%ZZDHCP%" =
"False"'>"1"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0EnableWINS" Condition='"%ZZDHCP%" =
"False"'>"True"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0WINSServerList" Condition='"%ZZDHCP%" =
"False"'>"10.10.1.200,10.10.5.200"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0DNSServerList" Condition='"%ZZDHCP%" =
"False"'>"10.10.1.1,10.10.5.100,10.10.8.50"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0EnableDNSRegistration" Condition='"%ZZDHCP%"
= "False"'>"True"'</Action>
    <Action Type="TSVar" Name="OSDAdapter0DNSDomain" Condition='"%ZZDHCP%" =
"False"'>"lab1.configmgrftw.com"'</Action>
```

These actions set various action task sequence variables with the static IP address information entered in Step 15 as well as other relevant information to support configuring a system with a static IP address. These variables are used by the Apply Network Settings task within the task sequence to populate the in use unattend.xml file which in then used by Windows setup to apply these settings.

18. `<Action Type="TSVar" Name="SMSTSUdaUsers" >"%ZZUDAUsers%"</Action>`

This action sets the SMSTSUdaUsers built-in task sequence variable.

19. `<Action Type="TSVar" Name="OSDComputerName" >"%ZZComputerName%"</Action>`

This action sets the OSDCOmputer built-in task sequence variable which in turn is used to name the deployed system by setting the value in the in-use unattend.xml.

20.
```
<Action Type="TSVar" Name="OSDBuildType" >"%ZZBuildType%"</Action>
    <Action Type="TSVar" Name=" OSDBuildType " Condition='"%ZZKiosk%" =
    "True"'>"%ZZKioskType%"</Action>
```

```
    <Action Type="TSVar" Name=" OSDBuildType " Condition='"%ZZKiosk%" =
"False"'>"STD"</Action>
    <Action Type="TSVar" Name=" OSDBuildType " Condition='"%ZZHomeSystem%" =
"True"'>"Home"</Action>
    <Action Type="TSVar" Name=" OSDBuildType " Condition='"%ZZDeveloper%" =
"True"'>"DEV"</Action>
```

These actiosn set various other task sequence variables that are used within the task sequence to determine other customizations and tasks to run.

```
21. </Actions>
    </UIpp>
```

These are the final closing tags: all elements in XML must have a closing tag or be closed.

9 CONFIGURATION FILE

The configuration file is XML based. The following table describes the valid elements. Note that UI++ does not check the configuration file against an XSD schema so will most likely ignore additional or out of place elements. This is not guaranteed though, so definitely test your configuration file outside of OSD and before you use it production. Keep in mind that XML is case-sensitive and that this format is not compatible with OSD++ 1.0 or OSD AppTree.

9.1 FEATURE CONFIGURATION & EXAMPLE SNIPPETS

9.1.1 Action Groups

2.10.0.0

Action Groups enable the grouping of other action elements into a single unit. There are two main reasons to group actions together: organization and conditional execution of grouped actions.

All Action Groups should have names specified using the Name attribute. Action Groups may or may not a have a Condition attribute that is evaluated. If an Action Group does have a condition attribute, sub-actions of this group will be skipped if this condition does not evaluate to true.

An action group can have any number of sub-actions and all Action types may be part of an Action Group.

```
<ActionGroup Name="Laptop Group" Condition="'%ChassisType%' = 'Laptop'">
  …
</ActionGroup>
```

9.1.2 AD Authentication

This action is used to present a user authentication dialog to the interactive user. It contains three fixed text input fields: User name, Password, Domain. The entered user name and password are authenticated against the domain specified (which should be in FQDN format).

If successful, the next action in the configuration XML is processed. Additionally, the user name is stored in the XAUthenticatedUser variable and domain name is stored in the XAuthenticatedUserDomain variable. Using the MaxRetryCount attribute, only a set number of failed authentication attempts is allowed after which an error message is displayed and the only option is to exit the dialog. This also returns an access denied (5) error code which can be validated in a task sequence using the _SMSTSLastActionRetCode task sequence variable or by checking the processes exit code when called using another method.

Multiple AD security groups can also be specified for this action. If specified, in addition to authentication, the user whose credentials are supplied must also be a member of one of the groups, specified in a semi-colon separated list, within AD. This provides an authorization mechanism in addition to the authentication.

Additionally, if authentication is successful, a task sequence variable named XAuthenticatedUserGroups can be populated with a list of all groups that the authenticated user is a member of. This is a comma separated list of groups in the usual <domain>\<group name> format. This can in turn be used as any other task sequence variable can be (within UI++ or the rest of the task sequence) including within conditions.

If desired, the prompt, question, and hint used for field as well as the regular expression can each be individually changed from their values using an embedded Field element (see the third example below):

Example:

```
<Action Type="UserAuth" Title="User Authentication" Domain="lab1.com" MaxRetryCount="5" Group="Test Group" GetGroups="True" />
```

Example of using XAuthenticatedUserGroups in a condition:

```
<Action Type="Info" Name="myInfo" Title="Welcome %XAuthenticatedUser%" Condition='InStr ("%XAuthenticatedUserGroups%" &amp;amp; ",", "Test,") > 0'>
<![CDATA[UI++ 2.0 includes all of the power of UI++ 1.0 combined with UI AppTree!<br>It's UI, <b>interactive </b>, evolved, and customized.<br>]]>
</Action>
```

Example of selectively customizing the prompt, hint, question, and regular expression of the fields in the UserAuth dialog:

```
<Action Type="UserAuth" Title="User Authentication" Domain="lab1.configmgrftw.com" Group="Test" GetGroups="False" MaxRetryCount="5">
<Field Name="Username" Prompt="Custom Username Prompt" Hint="Custom Username Hint" RegEx="[\w\-_.]+"/>
<Field Name="Domain" Question="Custom Domain Question" />
</Action>
```

2.9.3.0 Instead of using a text box for the domain name, a drop-down list can also be used to limit the choices given to the user. The following example shows a domain list.

```
<Action Type="UserAuth" Title="User Authentication" Domain="lab300.configmgrftw.com" GetGroups="False" MaxRetryCount="5" DisableCancel="False">
<Field Name="Domain" List="lab300.configmgrftw.com,lab200.configmgrftw.com" AutoComplete="true"/>
</Action>
```

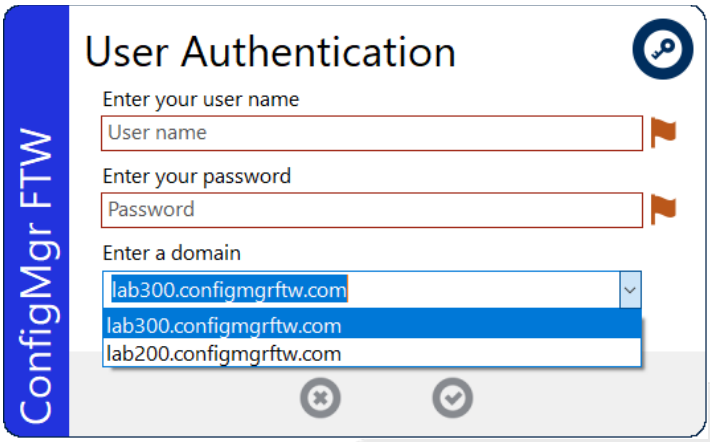


Figure 26: A user authentication dialog with a drop-down list of domains

2.10.3.0

9.1.2.1 User Attributes

By specifying a list of attributes (comma or semi-colon separated) using the **Attributes** attribute, the **UserAuth** action will query AD for the specified from the authenticated user's account and populate a variable for each. Each variable is named based on the name of the attribute: *XAuthUserAttribute_<attribute name>*. Non-existent attributes and attributes without a value are indistinguishable and neither populate a variable. Note that the attribute names displayed in ADUC are not necessarily the underlying attribute names – be sure to validate the real attribute names using the AD schema reference of the User class, ADSIEdit, the ADUC Attribute Editor, or some other equivalent method.

The following example collects the AD attributes *description* and *physicalDeliveryOfficeName* (which is labeled as just Office in ADUC) and populates the variables *XAuthUserAttribute_description* and *XAuthUserAttribute_physicalDeliveryOfficeName* with their values (respectively).

```
<Action Type="UserAuth" Title="User Authentication"
Domain="lab300.configmgrftw.com" MaxRetryCount="5" Group="ConfigMgr Techs"
Attributes="description,physicalDeliveryOfficeName"/>
```

9.1.3 AppTree

This action displays a customizable tree to the interactive user where they can choose packages and applications to install. Multiple AppTree actions can be used as necessary.

The software items are defined in two phases. The first is using either an **Application** or **Package** element within the top level **Software** element. These define the absolute properties for a specific software item including the label (also known as the display name) as well as the ConfigMgr Application name or the ConfigMgr package and program IDs. The Application name or package IDs must match exactly what is defined in ConfigMgr. The order of the **Application** and **Package** elements also defines the order in which the TS variables are created and thus their installation order. Note however that order is only relevant within either applications or packages as a ConfigMgr task sequence install all applications in one fell swoop and packages in one fell swoop.

Note that the top level **Software** element is a child of the **UIpp** element and thus a sibling to the **Actions** element.

The Id attribute specified must be unique for each **Application** or **Package** element and GUIDs are recommended although you can use any unique value as long as it unique within the configuration file. Visual Studio includes a GUID generator and there are also multiple available on the web. GUIDs can of course increase the difficulty of reading and deciphering the intent of the XML file though so consider this when selecting your standard unique IDs.

```
<UIpp Title="UI++" Icon="UI++2.ico">
     <Software>
         <Application Id="59426C81-2638-47AF-82A7-AFEA795B47B7" Label="Adobe Reader XI"
     Name="Adobe Reader XI"/>
         <Application Id="D30B903C-95ED-4AC9-8256-EFADD02FAFF3" Label="Notepad++ v6.6.8"
     Name="Notepad++ v6.6.8" />
         <Package Id="9EBF5537-6A81-4651-86D4-4E51C8899F4D" Label=".NET Framework 4.5.2"
     PkgID="ONE000100" ProgramName="Install .Net 4.5.2" />
         <Package Id="E6677316-BA46-4553-A8B8-0818875DFADB" Label="Internet Explorer 11"
     PkgID="ONE000101" ProgramName="Install IE11" />
         <Application  Id="7D2F6F33-38DA-404C-9E10-1A3845BE0270"  Label="Royal  TS  V2"
     Name="Royal TS V2" />
         <Application Id="9E30C625-9B5D-480D-AA55-6055F713AE29" Label="Microsoft Office
     2013" Name="Microsoft Office 2013" />
       </Software>

     <Actions>

     …

     </Actions>
</UIpp>
```

The second part is to define **SoftwareRef** and **SoftwareGroup** elements inside a **Set** element. **SoftwareGroup** elements define groups to show within the tree and can contain other **SoftwareGroup** elements as well as **SoftwareRef** elements. **SoftwareRef** elements are references to **Application** or **Package** elements and matched using the **Id** attribute. The order of **Application** and **Package** elements determines display order only and not application installation order.

```
<Actions>

      …

   <Action Type="AppTree" Title="Please choose your software">
     <SoftwareSets>
       <Set Name="Default">
         <SoftwareGroup Id="952025F7-BC5D-4D1C-960C-002B77323479" Label="Group A">
           <SoftwareGroup Id="07827D9D-8B57-444E-AC86-08D6DF527DC9" Label="Group B">
             <SoftwareRef Id="59426C81-2638-47AF-82A7-AFEA795B47B7" />
             <SoftwareRef Id="E6677316-BA46-4553-A8B8-0818875DFADB" />
           </SoftwareGroup>
           <SoftwareRef Id="7D2F6F33-38DA-404C-9E10-1A3845BE0270" />
           <SoftwareRef Id="9E30C625-9B5D-480D-AA55-6055F713AE29" />
         </SoftwareGroup>
         <SoftwareRef Id="D30B903C-95ED-4AC9-8256-EFADD02FAFF3" />
         <SoftwareGroup Id="71354335-19C7-4E12-A3D4-1B48EC91E7B4" Label="Group C">
           <SoftwareRef Id="9EBF5537-6A81-4651-86D4-4E51C8899F4D" />
         </SoftwareGroup>
       </Set>
     </SoftwareSets>
```

```
    </Action>
        ...
</Actions>
```

Software chosen by the interactive user will populate one of two task sequence variable bases – one for selected applications and one for selected packages – with a sequence number appended. This sequence number is two digits for applications and three digits for packages, starts at 01 or 001, and is incremented for each additional application or package selected. By default, the two base variables are XApplications and XPackages but these can be overridden using the **ApplicationVariableBase** or **PackageVariableBase** attributes. These base variable names should then be supplied to Install Application or Install Package steps that occur later in the task sequence during the Windows portion. For more details on using base variables for these task sequence steps, see the details section at http://technet.microsoft.com/en-us/library/hh846237.aspx#BKMK_InstallApplication and http://technet.microsoft.com/en-us/library/hh846237.aspx#BKMK_InstallPackage.

As just mentioned, this task does **not** install or initiate the installation of any software. It merely populates task sequence variables which are then later used by the appropriate and applicable task sequence steps to actually install the chosen software.

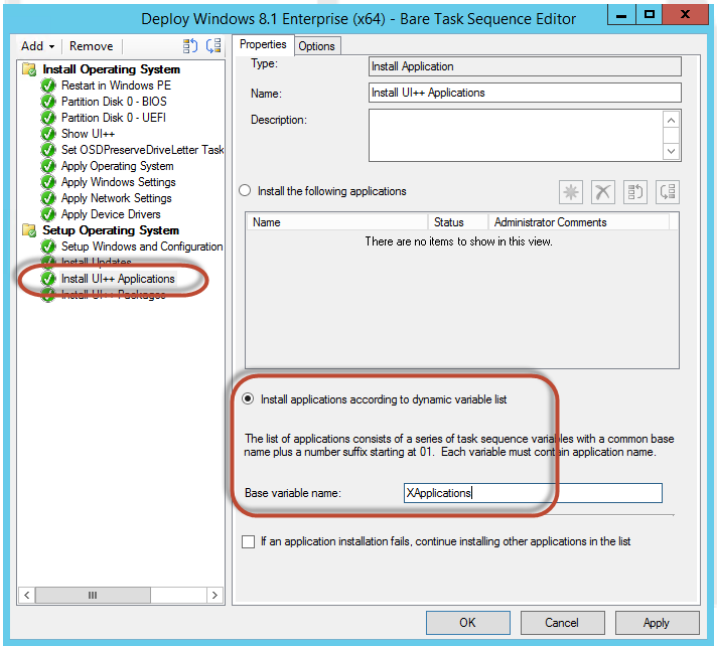


Figure 27: Initiating Chosen Application Installation

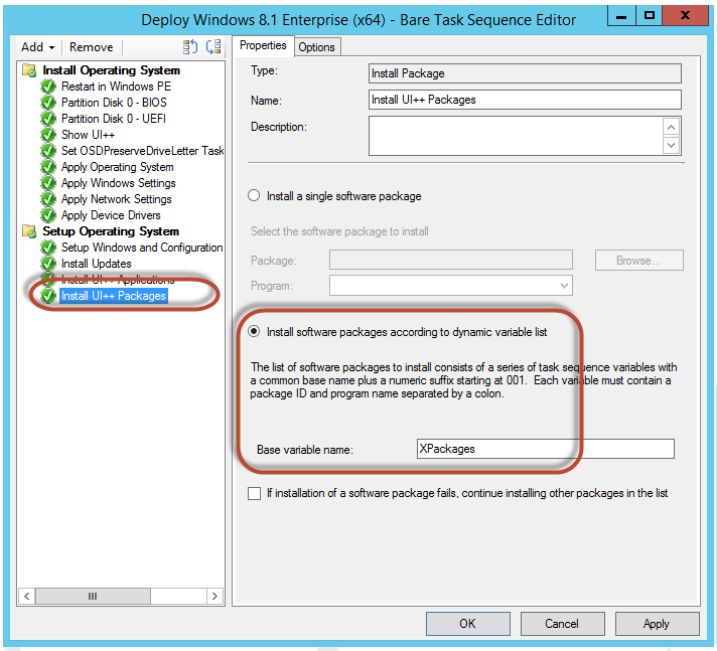


Figure 28: Initiating Chosen Package and Program Installation

9.1.3.1 Required Software and Groups

Software references or groups with the **required** attribute set to True are automatically selected and cannot be unselected. If a software group is set as required, all of its children are also set to required. Required groups and software in the tree will have their checkboxes greyed out and thus cannot be unselected. Additionally, required groups and software reference items will have their icon changed as listed in Table 1.

```
<SoftwareGroup Id="07827D9D-8B57-444E-AC86-08D6DF527DC9" Label="Group B" Required="True">
      <SoftwareRef Id="59426C81-2638-47AF-82A7-AFEA795B47B7" />
</SoftwareGroup>
```

In the above example, the group is set to required which in turn sets the child software reference to required also.

```
<SoftwareRef Id="59426C81-2638-47AF-82A7-AFEA795B47B7" Required="True"/>
```

In this example, the software reference is set to required.

9.1.3.2 Default Software and Groups

Software references or groups with the **required** attribute set to True are automatically selected; this is only the default state however and the user can uncheck this items if they desire. If a software group is set as default, all of its children are also set to default.

```
<SoftwareGroup Id="07827D9D-8B57-444E-AC86-08D6DF527DC9" Label="Group B" Default="True">
      <SoftwareRef Id="59426C81-2638-47AF-82A7-AFEA795B47B7" />
</SoftwareGroup>
```

In the above example, the group is set as default which in turn sets the child software reference to default also.

```
<SoftwareRef Id="59426C81-2638-47AF-82A7-AFEA795B47B7" Default="True"/>
```

In this example, the software reference is set as default.

Note that setting either a group or software reference to default when it is already required is meaningless.

9.1.3.3 Included Software

Applications and Packages can be set to include other Applications or Packages using the **IncludeId** attribute. Multiple ids can be included by separating them with a semi-colon. Included software is only processed for the first level. Thus, if A includes B and B includes C, A will not automatically include C. If this is needed, simply specify both B and C as included in A.

If a software reference is selected by the end-user in the tree, all software references corresponding to the applications and packages included in that application or package will also be selected. Additionally, software references set as required or default will also automatically select those same included applications and packages. Software references selected based on their inclusion in another selected item cannot be unselected by the end-user; these software references will have their checkbox greyed and their icon changed as listed in Table 1.

Unselecting a software reference with included items will also unselect those items unless they were previously selected by the user, were selected because they were default, or are required.

Included items do not truly model dependencies. This is because it is possible for an application to include a package and vice-versa. Depending upon how you add the corresponding Install Software and Install Application steps to the task sequence, either all selected applications will be installed first or all selected packages will be. Additionally, actual installation order or packages and applications as mentioned above is determined by the order that they are listed within the **Software** element.

```
<Application Id="59426C81-2638-47AF-82A7-AFEA795B47B7" Label="Adobe Reader XI"
Name="Adobe Reader XI" IncludeId="D30B903C-95ED-4AC9-8256-EFADD02FAFF3"/>
<Application Id="D30B903C-95ED-4AC9-8256-EFADD02FAFF3" Label="Notepad++ v6.6.8"
Name="Notepad++ v6.6.8" />
<Package Id="9EBF5537-6A81-4651-86D4-4E51C8899F4D" Label=".NET Framework 4.5.2"
PkgID="ONE000100" ProgramName="Install .Net 4.5.2" />
<Application Id="7D2F6F33-38DA-404C-9E10-1A3845BE0270" Label="Royal TS V2"
Name="Royal TS V2" IncludeId="59426C81-2638-47AF-82A7-AFEA795B47B7;D30B903C-95ED-
4AC9-8256-EFADD02FAFF3"/>
```

In the above example, the first application, Adobe Reader XI, includes the second, Notepad++. Additionally, the last application, Royal TS, includes the package for .Net as well as the Notepad++ application.

9.1.3.4 Hidden Software

Software references can be set to hidden by setting the **Hidden** attribute to True – groups cannot be set to hidden. Hidden software references are just that, not shown in the tree and thus cannot be selected or unselected by the end-user. Hidden software references can be set as required or default to automatically select them. Hideen software references do inherit the default or required state of their parent groups when the AppTree dialog is launched. They do not however toggle based upon the state of their parent. Hidden software references can be selected or unselected by setting them as included or excluded in Application or Package items (not software references).

```
<SoftwareRef Id="59426C81-2638-47AF-82A7-AFEA795B47B7" Hidden="True" Required="True"/>
```

This example shows a hidden software reference that is selected by default and thus will be included in the task sequence variables created.

9.1.3.5 Icons

Table 1 lists the various icons used in the AppTree, their meanings, and whether or not the user can toggle them.

Table 1: AppTree Icon Meanings

Icon	Item Type	Meaning	Manual State Change Allowed?
	Group	N/A	Yes
	Group	Required	No
	Software Reference	N/A	Yes
	Software Reference	Required	No
	Software Reference	Included	No
	Software Reference	Excluded	No

9.1.4 Default Values

The **DefaultValues** action can be used anywhere within the **Actions** tag but is typically best placed at the beginning as the first action. The below table (Table 2) shows all of the values set in this action. Note that the all begin with 'X' to avoid any conflicts with MDT variables (or any others folks may be creating). Many of these variables will contain the same information as MDT variables and in fact many extract data from the same locations and so may be redundant if you are using MDT. Because of the naming difference, there will be no conflicts however.

To retrieve a subset of the default values, specify the **ValueTypes** attribute along with the category of the values you wish to have retrieved per the categories listed in Table 2. Specify multiple categories to retrieve using a comma separated list. Omit the **ValueTypes** attribute or set it to **All** to retrieve all default values.

This example retrieves only TPM information:

```
<Action Type="DefaultValues" ValueTypes="TPM"/>
```

This example retrieves TPM, Asset, and OS information:

```
<Action Type="DefaultValues" ValueTypes="TPM,Asset,OS"/>
```

The following two examples retrieve all default values:

```
<Action Type="DefaultValues" ValueTypes="All"/>
```

```
<Action Type="DefaultValues" />
```

Table 2: Default Values

Variable	Description	Example Values	Category
XCPUArchitecture	The architecture of the CPU	x86, x64[1]	Asset
XCPULogicalCount	The number of logical CPUs in the system	8, 4, 2	Asset
XCPUNX	Whether the CPU supports No Execute (NX) functionality	True, False[1,2]	Asset
XCPUPAE	Whether the CPU supports Physical Address Extensions (PAE)	True, False[1,2]	Asset
XCPUSSE2	Whether the CPU supports SSE2	True, False[1,2]	Asset
XCPUVendor	The vendor of the CPU	GenuineIntel, AuthenticIntel[1]	Asset
XHWAssetTag	The asset tag of the hardware	Various based on the hardware vendor	Asset
XHWChassisType	The chassis type of the system	Desktop, Laptop, Server, Unknown[1]	Asset
XHWManufacturer	The manufacturer of the hardware	Various based on the hardware vendor	Asset
XHWMemory	The amount of physical memory in MB	16329, 4096	Asset
XHWModel	The model of the hardware	Various based on the hardware vendor	Asset
XHWProduct	The product name of the hardware	Various based on the hardware vendor	Asset
XHWSerialNumber	The serial number of the hardware from the BIOS	Various based on the hardware vendor	Asset
XHWUUID	The UUID of the hardware		Asset
XOnBattery	Whether the system is currently on battery or not	True, False[1]	Asset

[1] The values listed as examples are the only possible values for the variable

[2] PAE, NX, and SSE2 are required for Win 8 and WinPE 4.0

	XSystemUEFI	Whether the system is in UEFI mode or not	True, False[1]	Asset
	XSystemSecureBoot	Whether Secure Boot is enabled or not	True, False[1]	Asset
2.10.0.0	**XCurrentComputerJoinedToDomain**	If the system is currently joined to an AD domain	True, False[1]	Domain
	XCurrentComputerDomain	The current domain of the computer	Unbounded set.	Domain
	XCurrentComputerName	The current name of the computer	Unbounded set.	Domain
	XCurrentComputerOUDN	The current distinguished name of the computer's OU in Active Directory	Unbounded set.	Domain
2.10.0.0	**XComputerAzureDomainJoined**	Whether the system is Azure AD domain joined.	True, False[1]	Domain
2.10.0.0	**XComputerAzureDomainRegistered**	Whether the system is Azure AD registered by the current user.	True, False[1]	Domain
2.10.0.0	**XComputerAzureTenantId**[3]	The Tenant ID that the system is joined or registered to.	Unbounded set.	Domain
2.10.0.0	**XComputerAzureUser**[3]	The user that registered or joined the system to Azure AD.	Unbounded set.	Domain
2.10.0.0	**XComputerAzureDomain**[3]	The Azure AD name that the system is joined or registered to.	Unbounded set.	Domain
2.10.0.0	**XServiceStartModeSMSAgentHost**[3]	The start mode of the ConfigMgr agent service.	Auto, Manual, Disabled[1]	Mgmt
2.10.0.0	**XServiceStateSMSAgentHost**[3]	The state of the ConfigMgr agent service.	Stopped, Start Pending, Stop, Continue Pending, Pause Pending, Paused, Unknown[1]	Mgmt
2.10.0.0	**XConfigMgrSiteCode**[3]	The ConfigMgr site code that the agent is assigned to.	Unbounded set.	Mgmt
2.10.0.0	**XConfigMgrAgentVersion**[3]	The version of the ConfigMgr agent.	Various.	Mgmt
2.10.0.0	**XConfigMgrCurrentMP**[3]	The current MP that the ConfigMgr agent is assigned to.	Unbounded set.	Mgmt
2.10.0.0	**XMDMAuthorityName**[3]	The name of the MDM authority that the system is managed by.	Unbounded set.	Mgmt
	XIPGateway	A list of IP gateways for the system.	192.168.1.1	Net
	XMACAddress	A list of MAC Addresses for the system.	01:23:45:67:89:AB	Net
	XIPSubnetMask	A list of IP subnet masks for the system.	255.255.255.0	Net
	XWLANDisconnected	Whether the system is connected to a wireless network or not	True, False[1]	Net

[3] Not created or set if not applicable to the system.

	XWLANSSID	The SSID for the wireless network the system is connected to.	Unbounded set.	Net
2.9.2.0	**XWiredLANConnected**	If a wired network connection is detected.	True, False[1]	Net
	XOSArchitecture	The architecture of the OS	x86, x64[1]	OS
	XOSBuild	The build number of the OS	9200, 2600	OS
	XOSProduct	The OS product type	Workstation, Server[1]	OS
	XOSServerCore	Is the OS product running server core	True, False[1]	OS
	XOSServicePack	The OS (major[4]) service pack number	1, 2	OS
2.10.0.0	**XOSSystemDrive**	The system drive	C:	OS
	XOSVersion	The Windows OS version	6.2, 5.1	OS
	XOSWinPE	Is the OS WinPE – indicates whether the TS is currently in WinPE	True, False[1]	OS
	XOSCBSRebootPending[5]	A reboot is pending because of a CBS maintenance action.	True, False[1]	OS
	XOSWUARebootPending[5]	A reboot is pending because of a Windows Update action.	True, False[1]	OS
	XOSPendingFileRenameOperations[5]	A reboot is pending because of a file rename operation.	True, False[1]	OS
	XOSPendingComputerRename[5]	A reboot is pending because of a computer rename.	True, False[1]	OS
	XOSCCMRebootPending[5]	A reboot is pending because of a ConfigMgr action.	True, False[1]	OS
	XOSRebootPending[5]	A reboot is pending for any of the previous five reasons.	True, False[1]	OS
2.10.0.0	**XSystemDriveBitLockerProtected**[5]	Whether the system drive is protected by BitLocker	0, 1, 2^{16}	Security
2.10.0.0	**XServiceStateWindowsFirewall**	The start mode of the Windows Firewall service.	Auto, Manual, Disabled[1]	Security
2.10.0.0	**XServiceStartModeWindowsFirewall**	The state of the Windows Firewall service.	Stopped, Start Pending, Stop, Continue Pending, Pause Pending, Paused, Unknown[1]	Security
2.10.0.0	**XFirewallEnabledDomain**	Whether the Windows Firewall is enabled for the Domain Profile.	True, False[1]	Security

[4] To my knowledge, no OS service pack has ever used a minor number other than zero.
[5] Not populated in WinPE.
[6] See https://msdn.microsoft.com/en-us/library/windows/desktop/aa376483(v=vs.85).aspx#properties for a meaning of these values.

2.10.0.0	**XFirewallInboundDomain**	The blocking policy for inbound traffic for the Windows Firewall Domain Profile.	True, False[1]	Security
2.10.0.0	**XFirewallOutboundDomain**	The blocking policy for outbound traffic for the Windows Firewall Domain Profile.	True, False[1]	Security
2.10.0.0	**XFirewallEnabledPrivate**	Whether the Windows Firewall is enabled for the Private Profile.	True, False[1]	Security
2.10.0.0	**XFirewallInbounPrivate**	The blocking policy for inbound traffic for the Windows Firewall Private Profile.	True, False[1]	Security
2.10.0.0	**XFirewallOutboundPrivate**	The blocking policy for outbound traffic for the Windows Firewall Private Profile.	True, False[1]	Security
2.10.0.0	**XFirewallEnabledPublic**	Whether the Windows Firewall is enabled for the Public Profile.	True, False[1]	Security
2.10.0.0	**XFirewallInboundPublic**	The blocking policy for inbound traffic for the Windows Firewall Public Profile.	True, False[1]	Security
2.10.0.0	**XFirewallOutboundPublic**	The blocking policy for outbound traffic for the Windows Firewall Public Profile.	True, False[1]	Security
2.10.0.0	**XFirewallCurrentProfiles**	The currently active Windows Firewall profiles.	Public, Domain, Private	Security
2.10.0.0	**XServiceStateWindows DefenderAntivirusService**	The start mode of the Windows Defender service.	Auto, Manual, Disabled[1]	Security
2.10.0.0	**XServiceStartModeWindows DefenderAntivirusService**	The state of the Windows Defender service.	Stopped, Start Pending, Stop, Continue Pending, Pause Pending, Paused, Unknown[1]	Security
2.10.0.0	**XDefenderAVEnabled**	Whether Windows Defender anti-virus is enabled.	True, False[1]	Security
2.10.0.0	**XDefenderASEnabled**	Whether Windows Defender anti-spyware is enabled.	True, False[1]	Security
2.10.0.0	**XDefenderNISEnabled**	Whether Windows Defender network intrusion is enabled.	True, False[1]	Security
2.10.0.0	**XDefenderFullScanAge**	The number of days since the last full Windows Defender scan.	Number.	Security
2.10.0.0	**XDefenderAVSignatureAge**	The age of the Windows Defender signatures (in days).	Number.	Security
2.10.0.0	**XDefenderEngineVersion**	The version of the Windows Defender scan engine.	Various	Security

2.10.0.0	**XServiceStateWindowsUpdate**	The start mode of the Windows Update service.	Auto, Manual, Disabled[1]	Security
2.10.0.0	**XServiceStartModeWindowsUpdate**	The state of the Windows Update service.	Stopped, Start Pending, Stop, Continue Pending, Pause Pending, Paused, Unknown[1]	Security
2.10.0.0	**XWindowsUpdatesEnabled**	Whether Windows updates is enabled.	True, False[1]	Security
2.10.0.0	**XWindowsUpdateDefaultService**	What service Windows updates is using.	Various.	Security
2.10.0.0	**XWindowsUpdateServer**[3]	The WSUS server that Windows updates is using (if any).	Unbounded set.	Security
	XTPMEnabled[7]	Whether the TPM chip is enabled	True, False[1]	TPM
	XTPMActivated[7]	Whether the TPM chip is activated	True, False[1]	TPM
	XTPMAvailable[7]	Whether a TPM chip was detected	True, False[1]	TPM
2.10.0.0	**XTPMOwned**[7]	Whether the TPM chip is owned	True, False[1]	TPM
	XTPMSpecVersion[7]	The specification version of the TPM chip. See https://msdn.microsoft.com/en-us/library/windows/desktop/aa376484%28v=vs.85%29.aspx#properties for exact details of this attribute.	2.0, 0, 1.16	TPM
	XUserDisplayName[5]	The display name for current user from Active Directory or AzureAD.	Unbounded set.	User
	XUserSAMAccountName[5]	The SAM account name (including domain) for the current user.	Unbounded set.	User
	XUserPrincipalName[5]	The user principal name for the current user.	Unbounded set.	User
	XHWVirtualMachineType	Specifies the type of virtualization software used for the current systems	Hyper-V, VMWareESX, VMWareOther, VirtualBox, Xen, OtherMicrosoft[1]	VM

[7] Requires boot image of WinPE 4.0 or higher or the proper driver loaded in a full instance of Windows.

9.1.5 External Call

This action initiates the command-line specified. The example below calls the vbscript named QueryAD.vbs. What this script explicitly does is up to you. Location of the script is up to you but is best placed in the same package as UI++, in the boot image with UI++ (if using UI++ as a pre-start command), or in a separate package downloaded to the target system using the Download Package Content action in ConfigMgr Current Branch (1602 and above).

```
<Action Type="ExternalCall" Title="External Command">cscript.exe //NOLOGO //B QueryAD.vbs</Action>
```

Using an External Command action enables you to leverage existing scripts or other tools to perform a specific action that UI++ is not capable of. Possible activities for these external commands include calling a web service (which is much easier to do in PowerShell or VBScript), calling a driver installation routine, or initiating a reboot – many more possibilities of course exist.

If calling PowerShell from within WinPE, make sure that PowerShell is included in the WinPE boot image in use.

9.1.6 Error Info

The following snippet creates an error dialog box to inform the user of a fatal error or issue; it includes some minor HTML formatting. Note that use of <![CDATA[and]]> tags to enclose the text to display. These instruct the XML parser to ignore the contained content which is key when using any embedded HTML tags as these would try to the interpreted by the XML parser otherwise. Line breaks are added to wrap longer lines based upon spaces, periods, commas, forward slashes, and back slashes.

A cancel button is the only button shown on this dialog.

```
<Action Type="ErrorInfo" Name="SomethingBad" Title="Something went wrong>
      <![CDATA[I'm sorry, but the computer overlords are not happy with you today and this wizard cannot continue.<br><br>Have a nice day.]]>
</Action>
```

9.1.7 Info

The following snippet creates a user info dialog box; it includes some minor HTML formatting. Note the use of <![CDATA[and]]> tags to enclose the text to display. These instruct the XML parser to ignore the contained content which is key when using any embedded HTML tags as the XML parser would try to the interpret them otherwise. Line breaks automatically are added to wrap longer lines based upon spaces, periods, commas, forward slashes, and back slashes.

```
<Action Type="Info" Name="myInfo" Title="Welcome %Company%" Condition='"12" = "12"'>
      <![CDATA[UI++ 2.0 includes all of the power of UI++ 1.0 combined with OSD AppTree!<br>It's OSD, <b>interactive </b>, evolved, and customized.<br><br>Note that your current time zone is %TimeZone%.]]>
</Action>
```

2.9.0.0

A timeout can be set on an **Info** action so that it is automatically dismissed after the specified period. Upon automatic dismissal, you can configure the dialog to perform one of three activities:

- Act as if the user pressed the next button (which is the default) and move to the next action in the configuration file.

- Act as if it was cancelled by the user and cause UI++ to return an error code of 1223 (“The operation was canceled by the user”).
- Act as if it was cancelled by the user and cause UI++ to return a custom error code.

The countdown message displayed is also customizable if needed.

```
<Action Type="Info" Name="WelcomeInfo" Title=" Welcome" ShowCancel="True"
Timeout="120" TimeoutAction="Continue" >
      <![CDATA[Velkommen – Welkom - Tere tulemast – Bienvenue – Willkommen –
Bienvenido – Välkommen]]>
</Action>
```

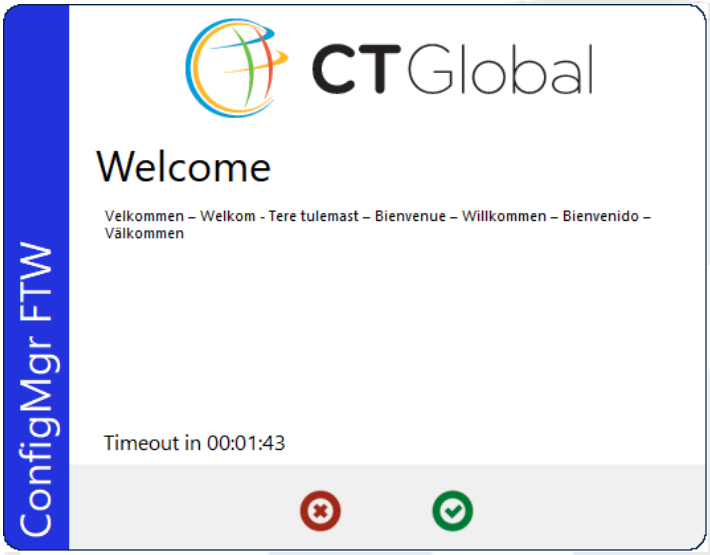


Figure 29: An Info dialog with a timeout

9.1.8 Input

The following snippet will create an **Input** dialog box with the specified information in the title:

```
<Action Type="Input" Name="myInput" Title="User Input Time" Size="Tall">
</Action>
```

Dialog boxes can contain as many input, drop-down list boxes, and info items that you need; however, only the first three, six, or nine will be displayed in the dialog box based upon the size configured. Using more than this number is valid though and can be useful when combined with conditions on the text or list boxes.

9.1.8.1 TextInput

A **TextInput** enables the user to enter text in response to a prompt or question. The value entered by the user is stored in a specified task sequence variable and can be validated using a standard regular expression.

The following example prompts the user to enter a desired name for the system and validates that the entered text is at least three characters long but no more than 15. The value entered is place in a task sequence variable named SystemName.

```
<Action Type="Input" Name="SystemNameInput" Title="Information" ShowBack="True">
```

```
      <TextInput Prompt="System Name" Hint="Please enter the desired name for this
system." RegEx=".{3,15}" Variable="SystemName" Question="Name of this system" />
    </Action>
```

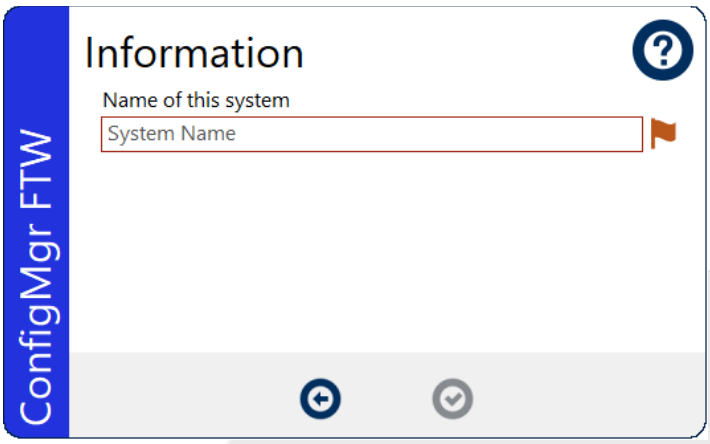


*Figure 30: An Input dialog with a single **TextInput***

9.1.8.2 ChoiceInput

This input type displays a drop-down list (also known as a combo-box, to the user from which they can choose a value in response to a question or prompt. Free text-entry is not allowed. A default value can be specified but is not required. Based on the user choice, the selected option or a representative value is placed in the specified task sequence variable. Optionally, an additional, alternate value can be placed in an alternate task sequence variable.

Choice sub-elements are used to define the choices presented in the drop-down list. Each **Choice** element defines the text shown in the list and a value and alternate value to populate the main and alternate task sequence variables if the user chooses this choice from the list. Specifying a value and alternate value are optional; if no value is specified, the actual value of the **Option** attribute is used to populate the main task sequence variable specified. If no alternate value is specified, then the alternate task sequence variable is not populated with any value.

The following example prompts the user to choose an OU for the system and based upon their choice populates the OUChoice and SystemPrefix task sequence variables.

```
    <Action Type="Input" Name="OUChoice" Title="AD Organizational Unit">
      <ChoiceInput Variable="OUChoice" AlternateVariable="SystemPrefix"  Sort="False"
Question="Please choose an AD OU for this system" Required="True" AutoComplete="False">
        <Choice Option="Finance"
Value="OU=Finance,OU=Workstations,DC=lab300,DC=configmgrftw,DC=com" AlternateValue="FIN"
/>
        <Choice Option="Human Resources"
Value="OU=HumanResource,OU=Workstations,DC=lab300,DC=configmgrftw,DC=com"
AlternateValue="HR" />
        <Choice Option="Information Technology"
Value="OU=IT,OU=Workstations,DC=lab300,DC=configmgrftw,DC=com" AlternateValue="IT" />
        <Choice Option="Manufacturing"
Value="OU=IT,OU=Workstations,DC=lab300,DC=configmgrftw,DC=com" AlternateValue="MAN" />
      </ChoiceInput>
    </Action>
```

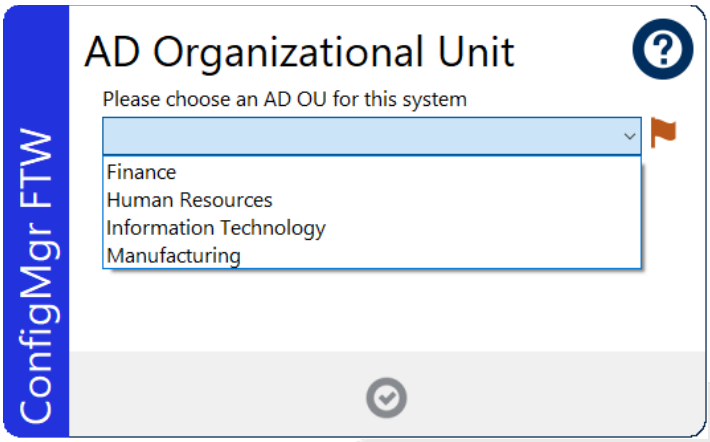


*Figure 31: An Input dialog with a single **ChoiceInput***

2.9.2.0

Instead of using multiple **Choice** sub-elements to specify each item in the list, a single (or multiple) **ChoiceList** sub-elements can be used. The **ChoiceList** element specifies comma-separated lists of items to add to the drop-down list as well as corresponding comma-separated lists of values and alternate values. This is useful to populate the list from an existing task sequence variable containing the list which in turn can be populated from a variety of sources including an external file.

The following example shows loading the options and alternate values for a **ChoiceInput** from two separate lists stored in task sequence variables. The resulting dialog is identical but the configuration is more concise and potentially flexible.

```
    <Action Type="TSVar" Variable="olist">"DCSS,Probation Adult and Juvenile,Agriculture
Department,ComDev,County Counsel,Tax Collector,Parks &amp;amp; Open Space,Cultural
Services,DA,Elections,Public Defender,Fire,Auditor,MCHHS,Human
Resources,IST,Assessor,Library Administration,County Administrators Office,Public
Administrator,DPW,Retirement,Board of Supervisors"</Action>
    <Action Type="TSVar"
Variable="alist">DCSS,PROB,AG,CDA,CC,DF,POS,CU,DA,ELE,PD,FIRE,DF,MCHHS,HR,IST,ARC,LIB,CAO
,PA,DPW,RET,BOS</Action>

    <Action Type="Input" Name="OUChoice" Title="AD Organizational Unit">
      <ChoiceInput Variable="OUChoice" AlternateVariable="SystemPrefix" Sort="False"
Question="Please choose an AD OU for this system" Required="True" AutoComplete="True">
        <ChoiceList OptionList="%olist%" AlternateValueList="%alist%" />
      </ChoiceInput>
    </Action>
```

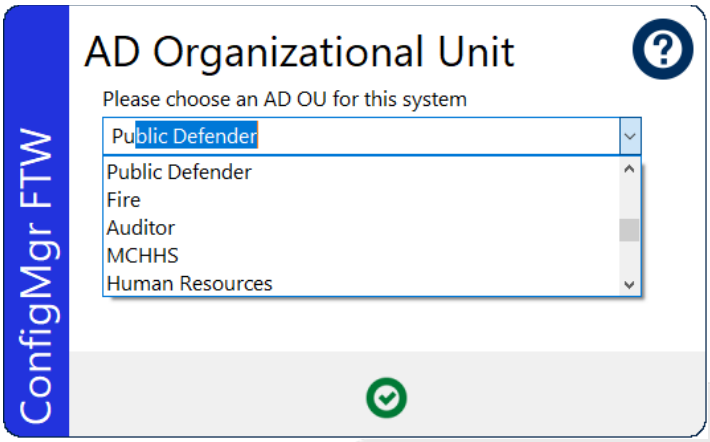


*Figure 32: An Input dialog with a single **ChoiceInput** populated using lists*

9.1.8.3 InputInfo

This type of input item on an **Input** dialog shows additional information to the user. It is completely free text, can be one or two lines, and a text color can be specified.

This example builds on the one from the TextInput section above and adds some additional info for the end user in the form of an **InputInfo** item.

```
    <Action Type="Input" Name="SystemNameInput" Title="Information" ShowBack="True">
      <TextInput Prompt="System Name" Hint="Please enter the desired name for this
system." RegEx=".{3,15}" Variable="SystemName" Question="Name of this system" />
      <InputInfo Color="#992233">Names must be between 3 and 15 characters
long.</InputInfo>
    </Action>
```

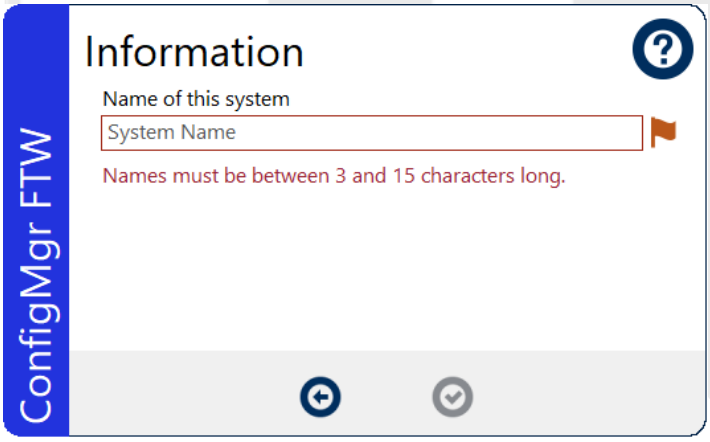


*Figure 33: An Input dialog with a single **TextInput** and a single **InputInfo***

9.1.8.4 CheckBoxInput

This final input type is a simple checkbox. It populates a specified task sequence variable with one of two values based upon whether the user checks the checkbox or not.

This example builds on the one from the TextInput section above and adds a checkbox. The task sequence variable DomainJoin will be populated with a value of Yes if the user checks the checkbox or NO if the checkbox is not checked.

```
    <Action Type="Input" Name="SystemNameInput" Title="Information" ShowBack="True">
      <TextInput Prompt="System Name" Hint="Please enter the desired name for this
system." RegEx=".{3,15}" Variable="SystemName" Question="Name of this system" />
      <InputInfo Color="#992233">Names must be between 3 and 15 characters
long.</InputInfo>
      <CheckBoxInput Variable="DomainJoin" Question="Should this system be domain
joined?" CheckedValue="Yes" UncheckedValue="No"/>
    </Action>
```

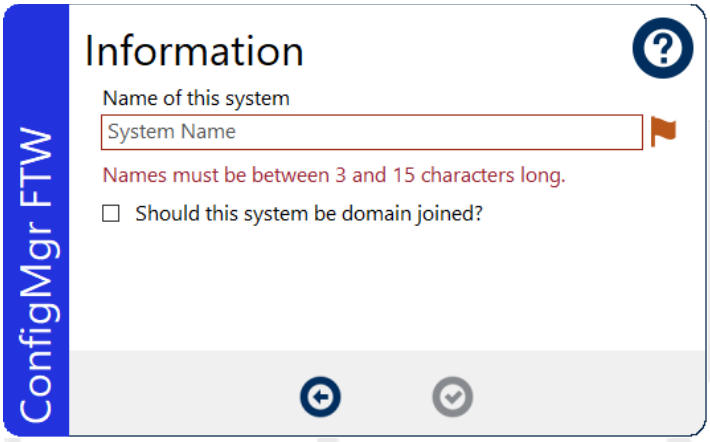


*Figure 34: An Input dialog with a single **TextInput** and a single **CheckboxInfo***

9.1.9 Files

This action does one simple thing: reads the first line from a text file and places the value of that line in a specified task sequence variable. It will also optionally delete that line from the text file. This is very useful for generating unique system names during OSD from a list of names, prefixes, or suffixes listed in that text file.

The following snippet reads the first line from the file named TextFile1.txt located in the root of the P drive, places the value of that line in the MyValue task sequence variable, and then deletes the line. The P drive of course needs to be accessible and the context in which UI++ is running must have read and write access to the file. This is easily done using a Map Network Drive task in the task sequence before UI++ is executed.

```
      <Action Type="FileRead" DeleteLine="True" Variable="MyValue"
      Filename="P:\TextFile1.txt" />
```

9.1.10 Preflight

This action performs a series of checks to ensure that the system is in a valid state before any additional action is taken. These checks are particularly useful to validate a system before a task sequence is run which causes major, irreversible changes to a system but may fail because of the current state of the system; e.g., being on battery power or connected to a wireless network.

Preflight checks are comprised of any valid VBScript expression and text to display for the check. The Preflight action displays a dialog box with each check listed (by its text) and the result of evaluating the expression. If any expression evaluates to false, the action as a whole fails and UI++ must be exited; this also results in UI++ returning a failure code of ERROR_NOT_READY (decimal 21).

The following is a standard preflight example that check for WLAN connectivity, whether a system is on battery, a minimum amount of memory, and whether the CPU supports Windows 8+ requirements.

```
    <Action Type="Preflight" Title="Preflight checks">
      <Check Text="WLAN Disconnected" CheckCondition='"%XWLANDisconnected%" = "True"' />
      <Check Text="Not on battery" CheckCondition='"%XOnBattery%" = "False"' />
      <Check Text="Minimum memory > 1GB" CheckCondition='%XHWMemory% >= 1024' />
      <Check  Text="CPU  Supports  Windows  8+"  CheckCondition='%XCPUPAE%  AND  %XCPUNX%  AND
%XCPUSSE2% = True' />
    </Action>
```

To add tooltips for the actual checks shown in the dialog, set the **Description** attribute for each check where a tooltip is desired. An example of this is shown in Figure 35.

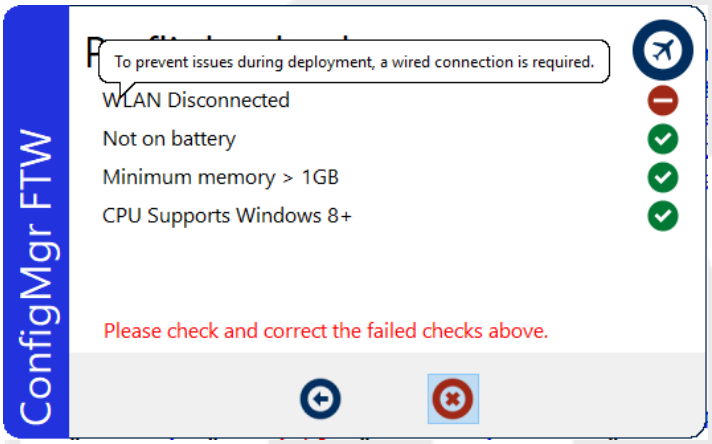


Figure 35: Preflight Check Descriptive Tooltip

To add tooltips to the check failed icon, set the **ErrorDescription** attribute for each check as desired. An example of this is shown in Figure 36.

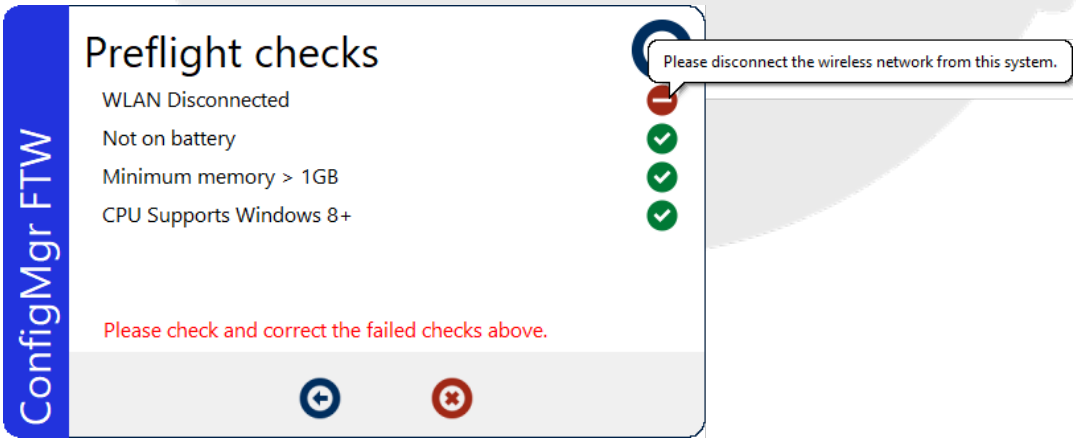


Figure 36: Preflight Check Descriptive Error Tooltip

9.1.11 SaveItems

This action saves or copies files to a specified location. There are five item types or files that it will save:

1. The SMSTS.log (including all previous smsts.log files that were created due to a rollover).
2. The UI++.log.
3. Any files in the current ConfigMgr agent log directory (%temp% in WinPE or if the ConfigMgr agent is not installed).
4. Any specified files.
5. A full dump of all current task sequence variables and their values.

The following snippet saves the UI++.log file, all smsts.log files, and a dump of all task sequence variables in a file called *Vars.txt* to the desktop of the user running UI++.

```
<Action Type="SaveItems" Items="UILog,TSVariables:Vars.txt,SMSTSLog"
Path="%userprofile%\Desktop" />
```

To specify additional log files from the existing ConfigMgr agent log location, add their file names to the **Items** attribute:

```
<Action Type="SaveItems" Items="SMSTSLog,BDD.log,Gather.log"
Path="%userprofile%\Desktop" />
```

Wildcards can also be used to copy multiple files as can full paths and environment variables:

```
<Action Type="SaveItems" Items="UILog,*.log,%windir%\Panther\*.log"
Path="%userprofile%\Desktop" />
```

The value of the **Path** attribute can be any valid UNC; however, keep in mind that the user running UI++ must have permissions to write to the UNC specified – UI++ can't magically overcome permissions. If run during OSD in ConfigMgr, this means that UI++ will run under the context of the local System account. If run after the **Setup Windows and ConfigMgr** task and the installed OS is joined to the domain, then the AD computer account of the computer will be used to access network resources. Adding a simple **Map Network Drive** task to the task sequence before running UI++ is a simple way to ensure access to any given UNC.

2.9.2.0

9.1.12 Switch

Similar to the SQL CASE statement and the C++ switch statement, the **Switch** action is an advanced way to populate the values of a single or multiple task sequence variables based on one of the following:

- A static value.
- The value of a task sequence variable.
- The value of an expression.

This value is then compared, in sequential order, to a series of regular expressions – the cases. The first regular expression to produce a match on the value sets the specified task sequence variables to the specified values. A default case can also be specified to set task sequence variables to default values when no cases are matched.

The following example sets the value of the TimeZone task sequence variable based upon the IP gateway detected by a **DefaultValues** action. There are four separate cases including a default case. The first case that matches the value is the one whose value is used to populate the task sequence variable.

```
<Action Type="Switch" OnValue='Trim("%XIPGateway%")' DontEval="False" >
  <Case RegEx="10\.0\.50\.1">
```

```
        <Variable Name="TimeZone">Pacific Standard Time </Variable>
      </Case>
      <Case RegEx="10\.127(\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){2}">
        <Variable Name="TimeZone">Central Standard Time</Variable>
      </Case>
      <Case
RegEx="(\W|^)(10\.35\.10\.1|10\.35\.11\.1|10\.36\.10\.1|10\.36\.11\.1)(\W|$)">
        <Variable Name="TimeZone">Romance Standard Time</Variable>
      </Case>
      <Default>
        <Variable Name="TimeZone">Eastern Standard Time</Variable>
      </Default>
    </Action>
```

9.1.13 Registry

This action either reads data from the registry storing it into a variable or writes data to it.

The following snippet reads the CurrentVersion value from the key SOFTWARE\Microsoft\Windows NT\CurrentVersion in the HKLM hive:

```
<Action Type="RegRead" Hive="HKLM" Key="SOFTWARE\Microsoft\Windows
NT\CurrentVersion" Value="CurrentVersion" Variable="CurrentVersion" />
```

The following snippet writes a string value named DefaultVal to the SOFTWARE\ConfigMgrFTW key in the HKLM hive:

```
<Action Type="RegWrite" Hive="HKLM" Key="Software\ConfigMgrFTW" Value="DefaultVal"
ValueType="REG_SZ">Nicky Romero</Action>
```

9.1.14 Task Sequence Variable

This action creates new variables with the specified name and value. It can be useful to create simple variables from static values or to set variable values dynamically.

The following snippet puts the static value “Finance” into the *BusinessUnit* variable.

```
<Action Type="TSVar" Name="BusinessUnit">Finance</Action>
```

You can also use the full power of VBScript to set the value of the variable.

```
<Action Type="TSVar" Name="LocationCode">Left("Amsterdam",5)</Action>
```

Note that the value of the variable is defined by the value of the element and not the value of an attribute. Also, you should typically embed the value of the element in a character data sub-element to avoid any XML parsing issues. This is shown in the example for the **Info** action next.

Additional note: The value specified is initially treated as a VBScript expression. If VBScript cannot evaluate the value, then the simple text of the value is placed in the variable; however, if the value is a valid VBScript expression, VBScript will evaluate it. Seemingly simple text like abc-def is actually valid VBScript (because the “-“ is seen as a minus sign) and the VBScript evaluation will thus result in the variable being set to zero. To avoid this, enclose the value in double-quotes (") if necessary:

```
<Action Type="TSVar" Name="OSDComputerName">"Finance-123"</Action>
```

9.1.15 Software Discovery

This action discovers software installed in the current Windows installation and sets a specified variable to **True** or **False** based on the existence of that software. Software is discovered from the **Add/Remove Programs** section of the registry and excludes system components; both 64-bit as well as 32-bit software is discovered on systems running 64-bit Windows but no explicit distinction is made between these. Discovery of software is first based on the display name of the software and then optionally on the version. Display name is matched using a case insensitive regular expression and version is matched using one of a number of operators as listed in Table 3. These operators properly evaluate standard version numbers; e.g., 1.2.3.4, 5.6.7 and 8.9. Note that version numbers with a greater number of sub-versions are greater than those with a less number less sub-versions if the first sub-versions are otherwise equal; e.g., 5.1.50428.0 is greater than 5.1.50428.

Table 3: Version Comparison Operators

Operator	Meaning
eq	Equals
gt	Greater than
gte	Greater than or equal to
lt	Less than
lte	Less than or equal to
ne	Not equal
re	Regular expression

The following example discovers the presence of three software items. The first is IIS Express based on a direct name match and a regular expression for the version. The second is Microsoft Silverlight using a regular expression for the display name and the gt operator for the version. The third is Snagit based on a regular expression name match; version is not considered. Each also sets the specified variable to either True or False based upon whether the discovery is successful or not. This variable can then be used in a Task Sequence Variable List action, an **AppTree** Action, or anywhere else this makes sense.

```
    <Action Type="SoftwareDiscovery">
      <Match DisplayName="IIS 10.0 Express" Version="10.0.\d{4}" VersionOperator="re"
Variable="IISExpressFound" />
      <Match DisplayName="Microsoft Silver\w*" Version="5.1.50428" VersionOperator="gt"
Variable="SilverlightFound" />
       <Match DisplayName="Snagit \d*" Variable="SnagItFound" />
    </Action>
```

9.1.16 Task Sequence Variable List

Using this Action, UI++ will create a sequence of variables populated with either application names or package IDs and program names. These lists of variables are suitable for use within either an Install Application or Install Package task within a task sequence.

The following snippet populates two lists: one for applications with **XApplicationsA** as its variable base for applications and one with **XPackagesA** for packages as its variable base. Applications and packages are specified by referring to the Id of the item defined earlier in the configuration XML using the **Software** element and **Application** and **Package** sub-elements as described in section **Error! Reference source not found.** (“**Error! Reference source not found.**”). This is the same section used by the **AppTree** action to

define software items. Applications and packages will be added to the correct list based upon their type and in the same order that they are specified within the **Software** element.

```
    <Action Type="TSVarList" PackageVariableBase="XPackagesA"
ApplicationVariableBase="XApplicationsA">
      <SoftwareListRef Id="7D2F6F33-38DA-404C-9E10-1A3845BE0270"
Condition='%IISExpressFound%'/>
      <SoftwareListRef Id="D30B903C-95ED-4AC9-8256-EFADD02FAFF3" />
      <SoftwareListRef Id="2D4090CA-DFE3-4434-9E3A-622FDC817965" Condition='Not
%SilverlightFound%'/>
      <SoftwareListRef Id="9EBF5537-6A81-4651-86D4-4E51C8899F4D" />
    </Action>
```

To control the content of the lists, use conditions. This is a perfect time to use the variables created by a **SoftwareDiscovery** action to dynamically build the list and create a mapping of software to install for refresh scenarios where you would like to restore software previously installed on a system.

9.1.17 Variable Saving and Loading

This action loads variables from a data file or saves the current variables to a data file. Saving only works if UI++ is executed outside of a task sequence; loading works within a task sequence or outside of it; if used to load variables within a task sequence, all variables and their values become task sequence variables available and manipulatable as all task sequence variables are. Read-only variables (those beginning with an underscore) and variables beginning with an 'X' are never saved or loaded.

This action is useful for persisting variables and their values for whatever reason between executions of UI++. The primary use case is to enable UI++ to be run before a refresh task sequence executed from Software center; this is usually done using a program that a task sequence depends upon. Doing this allows the UI++ interface to be shown without the use of ServiceUI.exe. Running UI++ in this way though prevents UI++ from saving variables as task sequence variables within the task sequence. Saving the variables during this initial execution of UI++ allows them to be re-loaded as the first step of a task sequence however.

To save the existing variables (set based upon user input and other actions during the execution of UI++) to a data file, add the following element as the last action within the UI++ configuration file:

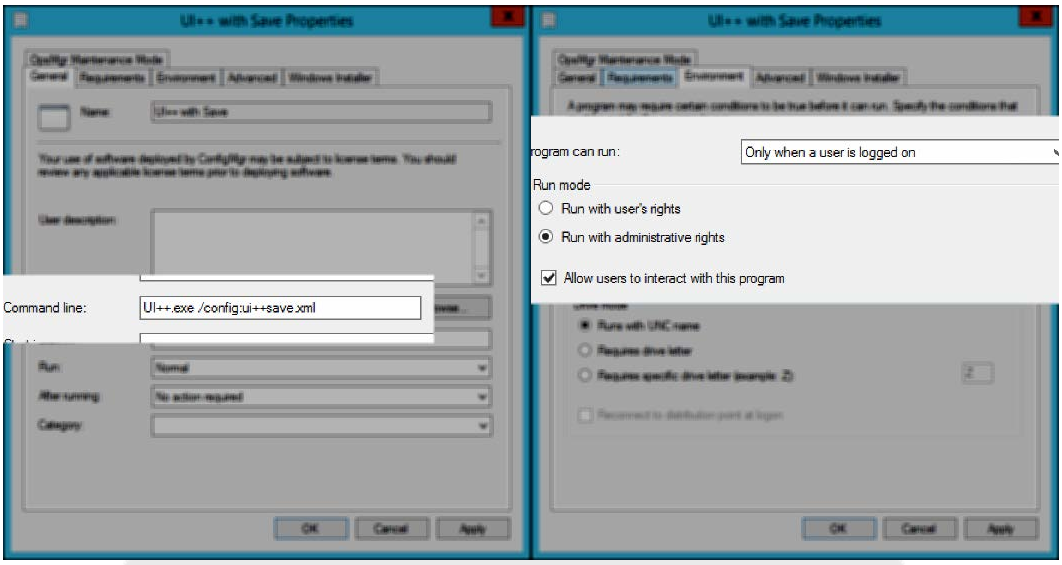
```
<Action Type="Vars" Direction="Save"/>
```

To load the variables, create a separate configuration file with the following element as the first action in a configuration file (other actions can be added as needed as well although remember that adding any user interface during a portion of the task sequence run from within Windows will result in no actual UI being shown to the interactive user and the task sequence hanging as it waits for user input based):

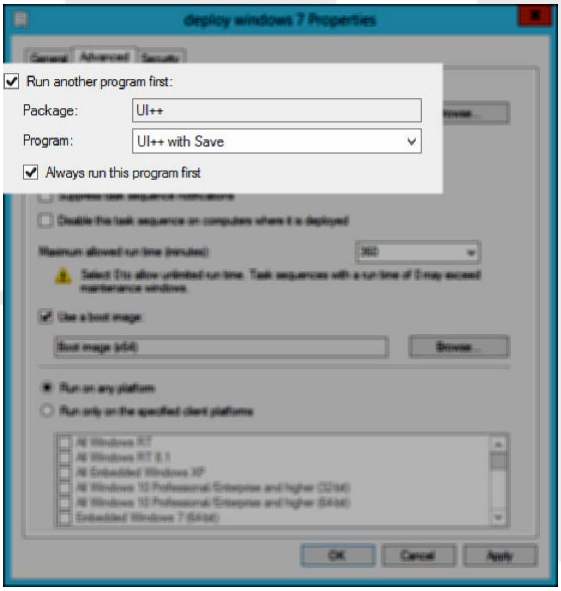
```
<Action Type="Vars" Direction="Load"/>
```

To use the above approach for a refresh task sequence, do the following.

1. Place both configuration files and UI++ into a package.
2. Create a program within this package to run UI++ using the first configuration file (use the /config parameter to specify the first configuration file). Make sure the program is set to run **Only when a user is logged on**, to **Run with administrative rights**, and to **Allow users to interact with this program** on the **Environment** tab of the program.



3. Create a second program to run UI++ without any user interaction that loads the variables saved by the first program. Use the /config parameter again to specify the second configuration file.
4. On the **Advanced** tab of the properties page for the refresh task sequence, select **Run another program first** and choose the package created in step 1 and program created in step 2. Select **Always run this program first**.



5. Within the refresh task sequence, at or new the beginning, add an **Install Package** task to run the second program created in step 3.

9.1.17.1 TextBox

Textboxes are for free form user input in response to a question or prompt. Text in an input box can validated against a regular expression. If no text is entered in the text box, a grey prompt will be shown if specified. The following table (Table 4) shows the cues shown to a user when the text entered does not match the regular expression.

If the variable specified for a text box already exists and contains a value, then this will be used as the default value for the text box; this will override any values specified using the Default attribute.

Table 4: Textbox Cues When text does not match Regular Expression

Has Focus	Cue
Yes	**Tooltip with hint text**
Both	**Text is red**
Both	**Red exclamation mark is shown after the text box**
No	**Textbox border is colored red**

The following snippet creates a text box prompting the user for the name to assign to the computer, validates it against a regular expression, and assigns the value to the *CompName* variable. An optional hint is displayed as a tooltip if the current input does not match the regular expression and an optional prompt is also displayed when the text box is empty.

```
<TextInput Question="Please enter a computer name" Regex="[A-Z]\d{3}.*"
Variable="CompName" Hint="Please enter an uppercase letter followed by three
digits." Prompt="Computer Name" Default="%OSDComputerName%"/>
```

Notice the use the %OSDComputerName% variable in the Default attribute; this automatically sets the default value of this text box to the value stored in the OSDComputerName task sequence variable at the time UI++ is executed which is the system's current name if UI++ is run in a refresh task sequence.

Regular expressions are beyond the scope of this documentation but many excellent articles and tutorials are available on the web including http://www.regular-expressions.info/reference.html.

9.1.17.2 Drop-Down list Box

List boxes provide a drop-down for the user to choose from based on a question or prompt.

The following snippet creates a list box prompting the user to choose a business unit and putting the value in an UI++ variable named *BusinessUnit*.:

```
<ChoiceInput Question="Are you in the finance department?" Variable="finance"
Required="True">
</ChoiceInput>
```

The initial value chosen in the list box is set using the **Default** attribute or is blank. If the **Required** attribute is set to "True" and no **Default** value is specified, then a choice must be made in the list box by the user for the OK button to be enabled in the dialog. If no **Default** value is specified and **required** is set to "False", then a red exclamation will be shown after the list box in the dialog.

If the variable specified for a ChoiceInput already exists and contains a value, then this will be used as the default value for the drop-down list box; this will override any values specified using the Default attribute.

The choices presented in the list box are defined by **Choice** sub-elements. Based on the user's selection, the value of that choice is placed in the variables named using the **Variable** attribute. The following snippet shows two simple options:

```
<Choice Option="Yes" Value="TRUE" />
<Choice Option="No" Value="FALSE" />
```

For list boxes with many possible choices, you can enable auto-completion using the **AutoComplete** attribute. Setting this to **True** enables the user to type into the edit box of the list box. This will be automatically completed based upon the available choices. If the text entered by the user does not match any available choices, then no choice is selected and the user cannot continue until they select or enter a valid choice.

9.1.17.3 Info

Info items provide a way to add arbitrary information or instructional text to an **Input** dialog. These Info items can be one or two lines of text (one is default); you can customize the text color (using standard web, RGB hex notation) and text will automatically wrap to the next line if necessary and if possible. You can add a line-break by adding "\r\n" to the value specified.

The following snippet creates an Info item with the color Dark Blue and includes two lines of text:

```
<InputInfo Color="#00008B">Make it so.</InputInfo>

<InputInfo NumberofLines="2" Color="#00008B">Resistance is futile.\r\nYou will be
assimilated.</InputInfo>
```

9.1.18 WMI

The following snippet queries WMI for the model of the system.

```
<Action    Type="WMIRead"    Namespace="root\cimv2"    Class="Win32_ComputerSystem"
Property="model" Variable="csmodel" />
```

The following snippet queries WMI for the state of the Windows Update Service.

```
<Action    Type="WMIRead"    Variable="WUAUServiceState"    Namespace="root\cimv2"
Class="Win32_Service" KeyQualifier='Name="wuauserv"' Property="State"/>
```

The following snippet opens or creates the root\ITlocal WMI namespace. If then creates the Local_Config WMI class if it does not exist. Finally, it creates or updates an instance of the Local_Config class by setting the ComputerName and Tier values as specified. If an object already exists with the same key property value, then this will overwrite the values in that object.

```
<Action Type="WMIWrite" Namespace="root\ITLocal" Class="Local_Config" >
      <Property   Name="ComputerName"   Type="CIM_STRING"   Value="%ComputerName%"
Key="True"/>
      <Property Name="Tier" Type="CIM_UINT8" Value="%Tier%" Key="False"/>
</Action>
```

9.2 The Back Button

A recent addition to UI++ is the ability to move backwards in the sequence of dialogs shown. You can selectively add the back button to each UI Action by adding the **ShowBack** attribute and setting it to True on an action that displays a dialog. For example, enabling the back button on an Input action would look something like this:

```
<Action Type="Input" Name="ClientSetupInput" Title="Client Setup" ShowBack="True">
```

The valid UI actions where you can add a back button include the following (and only the following):

- AppTree

- ErrorInfo
- Info
- Input
- PreFlight
- UserAuth

If a UI action is the first dialog shown, the back button will not be shown on the action's dialog regardless of the value of the **ShowBack** attribute.

Going back will preserve any of the user's previous entries or selections with a few exceptions:

- For **UserAuth** actions, only the domain will be preserved.
- For **PreFlight** actions, there's nothing to preserve and all checks will be re-run.
- For **AppTree** actions, all occurrences of a selected software item will be selected; i.e., if the tree shows an application or package multiple times in different locations and the user chooses one, if they then go back to the AppTree, all occurrences of that item will now be selected.

Going back jumps directly to the previous UI action that was shown and does not execute any intermediary actions including UI actions that were not shown because a condition prevented them from being shown. Going forward, all processing happens normally as if then back button was never pressed. User entries and selections will be preserved with the same exceptions as noted above. Keep in mind though that if a user enters some info on a dialog or makes a selection without pressing the OK button, those selections will never be preserved.

If you want the **Preflight** action to re-check the state of the system based upon values generated by the **DefaultValues** action, ensure that there is a UI action before the **Preflight** action. Place the **DefaultValues** action before the **PreFlight** action but after this other UI action. As noted in the previous item, the **DefaultValues** action will then be evaluated (or re-evalauted) every time the user goes forward from this UI action thus forcing the values to be re-detected and repopulated.

9.3 Values and Variables

When run within a task sequence, all values are stored directly as task sequence variables and thus usable within the task sequence after UI++ exits. Additionally, all valid task sequence variables are directly usable from within UI++. Note that this is a change from version 1 where variables were initially stored internally to UI++ only and had to be explicitly written out as a task sequence variable.

If UI++ is run outside of a task sequence, then UI++ will transparently use an internal variable system. This variable system is equivalent to the task variable system except that it is not available outside of a single instance of UI++. If you need to persist these values for use in a future instance of UI++, you can easily add them to the registry or WMI and then reload them.

Task sequence variables and internal variables can be directly used or referenced in any field within the XML configuration file. UI++ will replace any occurrence of a variable name surrounded by percent signs in the configuration file with the actual value of that variable. Thus is you have a variable named **CurrentVersion** and you define the following in the title attribute for a dialog, "Upgrade from Windows %CurrentVersion%", UI++ will replace %CurrentVersion% with whatever value **CurrentVersion** represents.

To dump a complete list of variables and their values from with UI++, simply press Ctrl+F3 on any dialog. This will create a text file named **UI++ Variable Dump <Date> <Time>.txt** in the same place that the UI++ log file is currently being written to. If the **/disabletsvareditor** switch is specified on the command-line, dumping the variables in this way to a file will be disabled.

9.3.1 Variable Replacement

For any attribute or value of any element in the configuration file, you can insert the value of any environment variable or task sequence variable (including anything that UI++ previously created or retrieved) by inserting the name of the variable surrounded by percent signs (%). If an environment variable and task sequence variable have the exact same name, the environment variable will take precedence.

The examples in section 9.4 ("Conditions") clearly show variable replacement for conditions. This same technique can be done for any attribute in any of the tags.

9.3.2 Boolean Variable Negation

For any attribute that expects a Boolean value; i.e., **True** or **False**, prepending the value with an exclamation point will negate the value specified. For example, the following adds a software item to an AppTree:

```
<SoftwareRef Id="E6677316-BA46-4553-A8B8-0818875DFADB" Default="!%SnagitFound%"/>
```

The **Default** attribute for this example will evaluate to the opposite of the value of the SnagitFound variable. Thus if SnagitFound is **True**, the opposite value, **False**, is instead used and vice-versa of course. If SnagitFound isn't a valid Boolean value, then **False** is passed.

Boolean variable negation is not applicable to Condition attributes. For Conditions attributes, simply use the VBScript **Not** operator.

9.4 Conditions

Every action along with many of the other configuration elements that are within the configuration file can be executed conditionally enabling for the actions as well as the look and feel to be dynamically customized at runtime. These conditions can be based on any valid VBScript expression and generally will be based on the values of task sequence variables. Because conditions are based on VBScript all of the functions of VBScript are available including normal string processing and math functions.

To add a condition to an element, simply add a **Condition** attribute. Conditions are valid for all elements listed in Table 5Table 6: Valid XML Elements. If the condition specified evaluates to false, than whatever the element represents will be skipped by UI++.

The following table shows the result of a false condition on each of the different type of elements.

Table 5: Valid Conditional Elements and False Condition Evaluation Effects

Element	Effect
Action	Action is completely skipped.
Check	The check is not performed.
Choice	Choice not added to the list box.

ChoiceInput	List Box not added to input dialog.
DefaultValues	Default values are not discovered or populated.
Match	No software discovery for the item is performed.
RegRead	Registry value is not written.
RegWrite	Registry value is not read.
Set	The software set and any contained groups or software references are not used to populate the AppTree.
SoftwareGroup	The software group and any other groups and software references it contains are added to the AppTree.
SoftwareRef	The software reference is not added to the tree or the variable list.
TextInput	Text box not added to the input dialog.
WMIRead	WMI value is not read.
WMIWrite	WMI value is not written.

Here are a few example conditions:

```
<Action Type="TSVar" Name="Company" Condition='"%OSDComputerName%" = "TheBoss"'>Acme, Inc.</Action>

<Choice Option="Maybe" Value="MAYBE" Condition='Len("%FirstAnswer%") = 5'/>

<TextInput Variable="FourthAnswer" Condition='"%SecondAnswer%" = "Dos"' Question="What is your fourth answer?" />
```

Notice that the condition's value is defined within single quotes instead of double quotes. This is perfectly valid for XML and is required because VBScript surrounds string literals with double-quotes. Notice also that the second example uses a VBScript function, Len, to evaluate the condition.

Table 6: Valid XML Elements[8]

Element Name	Valid Attributes	Valid Parents	Valid Children	Comments
Action	**Type*** – Specifies the type of action to perform.	Actions	Various, depends upon the Action type.	The different actions types are their valid attributes are described in Table 7.
ActionGroup	**Name** – The name of the group.	ActionGroup	Action ActionGroup	Groups actions into a single unit.
Actions	**N/A**	UIpp	Action	Groups all of the actions performed during UI++. Each child Action element is performed in the order that it occurs.
Application	**Id*** – A unique identifier for the application. This Id value is referenced by a SoftwareRef element to display the Application in the AppTree action. This can be any identifier you want it to be including a GUID as long as it's completely unique from other unique identifiers with the configuration XML. **IncludeId** – A semi-colon separated list of Application and Package IDs that are included when this Application is chosen. This can include hidden applications and packages. **Label*** – The label shown for the application in the AppTree action.	Software	N/A	This defines the properties for an Application that can be displayed to the interactive user in an AppTree action.

[8] Attributes denoted with an asterisk (*) in Table 6 and Table 7 are required. All other attributes are optional.

	Name* – The actual name of the Application as defined in ConfigMgr. This must match exactly including spelling and case.			
Case	**CaseInsensitive** – Performs a case insensitive match. Default is **False**. **DontEval** – If set to True, do not pass the value of the element to VBScript for processing. Default is false. **RegEx*** – The regular expression to compare against the value of the **OnValue** attribute of the parent **Switch** element.	Action (where Type = Switch)	Variable	The actual value set if the regular expression comparison succeeds is the value of this element and not an attribute.
Check	**CheckCondition*** – The definition of the condition to be checked. This can be any valid VBScript expression and can include task sequence variables. **Text*** – The text to display describing the check that was performed.	Action (where Type = Preflight)	N/A	Adds a check to a Preflight action.
Choice	**Option*** – Defines the text displayed in the drop-down list box. **Value** – The actual value assigned to the variable when this choice is selected. If not specified, the value of the Option attribute is used. **AlternateValue** – A second value assigned to the variable designated by the **AlternateVariable** attribute of the parent **ChoiceInput** element if this choice is chosen by the interactive user.	ChoiceInput	N/A	Adds a choice to the drop-down list box.
CheckboxInput	**CheckedValue** – The value to store in the specified variable if the checkbox is checked. Default is True. **Default** – The default value for the checkbox. If this value matched the specified CheckedValue, then the checkbox will be checked. **Question*** – The question to display above the drop-down list box. **Variable*** – The variable name to store the value of the chosen option in.	Action (where Type = Input)	N/A	Adds a checkbox to an input dialog.

2.9.2.0

	UnheckedValue – The value to store in the specified variable if the checkbox is unchecked. Default is False.			
ChoiceInput 2.10.1.0 2.9.1.0	**AlternateVariable** – The variable name to store the alternate value of the chosen option in. **AutoComplete** – Enables automatic completion of the text entered in the edit box portion of the combo box. Default is false. **Default** – The default choice in the list box. **DropDownSize** – The maximum size of the drop-down list; i.e., the maximum number of items shown in the drop list at a single time. If there are more items in the list, the drop-down will contain a vertical scrollbar. Default is 5. **Question*** – The question to display above the drop-down list box. **Required** – Whether or not a value must be selected. **Sort** – Sorts the items in the drop-down list box. Default is True. **Variable*** – The variable name to store the value of the chosen option in.	Action (where Type = Input)	Choice	Adds a drop-down list box to an input dialog.
2.9.2.0 **ChoiceList**	**AlternateValueList** – A comma-separated list of values used to populate the **AlternateVariable** specified in the parent **ChoiceInput** element based upon the choice made by the user. **OptionList** – A comma-separated list of values to add to the drop-down list created by the parent **ChoiceInput** element. **ValueList** – A comma-separated list of values used to populate the **Variable** specified in the parent **ChoiceInput** element based upon the choice made by the user.	ChoiceInput	N/A	Adds a list of items to the drop-down list from a comma-separated list of values.
Field	**Name*** – The UserAuth dialog field to affect; must be one of “Username”, “Password”, or “Domain”. **Hint** – The custom message to display in the tooltip shown when the text entered does not match the regular	Action (where Type = UserAuth)	N/A	

2.9.3.0	expression; defaults are “Please enter your user name”, “Please enter your password”, and “Please enter a domain”. **List** – This attribute is only valid for the Domain field. If specified, this creates a drop-down list instead of a plain text field that enables the user to choose a valid domain. This list should be comma-separated. The value of the **Domain** attribute of the parent **Action** element is used as the default value. If this attribute has no value, then no default value will be used and the user must choose a value. **Prompt** – The custom prompt text to display within the text box when it is empty; defaults are “User name”, “Password”, and “Domain”. **Question** – The custom question to display above the input box; defaults are “Enter your user name”, “Enter your password”, and “Enter a domain”. **ReadOnly** – For Domain fields only, setting this to true will prevent the user from changing the value. Default is false. **RegEx** – The custom regular expression to use to validate the text entered into the text box; defaults are “[^\"/\\\[\]:;|=,\+*\?<>]{3,15}”, “.+”, and “[\w\-_.]+”.			
InputInfo	**Color** – The hexadecimal color to display the text in. Defaults to #000000 (black). **NumberofLines** – The number of lines of text to be displayed. Default is 1.	Action (where Type = Input)	N/A	Adds info text to an Input dialog.
Match	**DisplayName*** – The display name to match when searching for installed software. Can be a regular expression. **Variable*** – The variable to store the result of the software search in. Value will be **True** or **False**. **Version** – The version of the software to match against.	Action (where Type = SoftwareDiscovery)	N/A	Specifies the criteria for finding installed software.

	VersionOperator – The operator to use when comparing versions. Default is eq (equals).			
Package	**Id*** – A unique identifier for the package. This Id value is referenced by a SoftwareRef element to display the Package in the AppTree action. This can be any identifier you want it to be including a GUID as long as it's completely unique from other unique identifiers with the configuration XML. **IncludeId** – A semi-colon separated list of Application and Package IDs that are included when this Package is chosen. This can include hidden applications and packages. **Label*** – The actual program name of the program within ConfigMgr. This must match exactly including spelling and case. **PkgId*** – The actual package ID of the package within ConfigMgr.	Software	N/A	This defines the properties for a Package and Program that can be displayed to the interactive user in an AppTree action.
Property	**Key*** – Whether the property is a key property of the class. Must be "true" or "False". **Name*** – The name of the property. **Type** – The WMI type for the property. Can be any valid WMI type including CIM_STRING or CIM_UINT8. CIM_STRING is the default if not specified. **Value*** – The value to assign to the property.	Action (where Type = WMIWrite)	N/A	Defines the properties to write to or create if a new class is being created.
Set	**Name*** – The name of the Set.	SoftwareSets	SoftwareGroup SoftwareRef	Contains groups and software to display to the interactive user in an AppTree action.
Software	**N/A**	UIpp	Application Package	This software element contains the applications and packages that an AppTree action can reference and display

				to the interactive user.
SoftwareGroup	**Default** – Automatically selects this group and all of its children when the dialog is created. Default is false. **Id*** – A unique identifier for the group. **Label*** – The label shown for the group in the AppTree action. **Required** – Same as default but the group and its children cannot be unselected. Default is false.	Set SoftwareGroup	N/A	A group to display in an AppTree action. Groups can contain other groups or software.
SoftwareListRef	**Id*** – The Id of the software item previously defined in the configuration XML file.	Action (where Type = WMIWrite)	N/A	A reference to a software item previously defined within the Software element. These are added to a task sequence variable list.
SoftwareRef	**Hidden** – This software reference is hidden from the tree. It will be selected if it is set as default or required (or its parent group is set to default or required) or if it is included by an application or package that is also selected. Note that unselecting a group that contains a hidden software reference does not in turn unselect that hidden software reference. Default is false. **Id*** – The Id of the software item previously defined in the configuration XML file. **Default** – Automatically selects this software reference when the dialog is created. Default is false. **Required** – Same as default but the software reference cannot be unselected. Default is false.	Set SoftwareGroup	N/A	A reference to a software item previously defined within the Software element. These are shown to the interactive user in the AppTree action.
SoftwareSets	**N/A**	Action (where Type = AppTree)	Set	A container for Set elements
Text	**Type** – The **DefaultValues** value type that the progress text applies to.	Action (where Type = DefaultValues)	N/A	If not specified,

	Value – The text to display in the progress dialog for the specified value type.			
TextInput	**Default** – The default value of the text box. **ForceCase** – Force the case of entered text to upper or lower by setting this attribute **Upper** or **Lower**. Default is to not force the case at all. **Hint** – The message to display in the tooltip shown when the text entered does not match the regular expression. **Password** – Hides the user input similar to entering a password. Default is false. **Prompt** – The prompt text to display within the text box when it is empty. **Question*** – The question to display above the input box. **RegEx** – The regular expression to use to validate the text entered into the text box. **Required** – Whether or not a value must be selected. Valid values include True, False, Yes, or No. If not specified, this defaults to true. **Variable*** – The variable name to store the resulting text in.	Action (where Type = Input)	N/A	Adds a text input box to an input dialog.
UIpp 2.10.3.0 2.9.1.0	**AlwaysOnTop** – Configured the UI++ dialogs to always on top of all other top-level windows. Default is True. **Color** – The hexadecimal color of the sidebar used in all dialogs. Defaults to #002147 (The University of Michigan blue). **DialogIcons** – Shows or hides the icons in the upper right corner of all dialogs. Default is to display the icons (true). **Flat** – Shows dialogs using a flat look and feel. Default is false. **Icon** – The icon to display in every dialog box shown. **Title** – The title used in the sidebar of every dialog box shown.	N/A	Apps Actions	This is the root element of the file.
Variable	**Name*** – The variable name to set a value for.	Case	N/A	This action statically sets a variable; the

2.9.5.0

	DontEval – If set to True, do not pass the value of the element to VBScript for processing. Default is false.	variable could be any pre-existing variable or a new one. The actual value set is from the value of this element and not an attribute.

The following table describes the different types of actions available. These are configured using **Action** elements with various **Type** values.

Table 7: Valid Actions[8]

Type	Valid Attributes	Comments
AppTree 2.9.4.0	**ApplicationVariableBase** – The base variable to populate with selected applications. Defaults to XApplications if not specified. **PackageVariableBase** – The base variable to populate with selected packages. Defaults to XPackages if not specified. **ShowBack** – Shows a back button on the dialog enabling the interactive user to return to the previous GUI based action. Default value is false. **ShowCancel** – Shows a cancel button allowing the user to gracefully exit UI++ without proceeding. Default is false. **Title** – The title of the user authentication dialog. **Size** – The height of the AppTree dialog displayed. Valid Options include Regular, Tall, and ExtraTall. Default is Regular. **Expanded** – Sets whether the tree is expanded or not when the action is first launched. Default is True.	Displays a dialog with a tree of selectable applications and packages.

DefaultValues	**ShowProgress** – Shows a dialog with a progress bar so that the user is aware of the background activity taking place. Possible valid values are **True** and **False**. Default value is **True**. **ValueTypes*** – The category of values to retrieve. Valid options include Asset, OS, TPM, Net, Domain, and All. To retrieve multiple categories of values specify a comma separated list of categories. Default value of the attribute is **All**.	Discovers and populates default variables based on the current system state. [9]
ErrorInfo	**Name** – The name of the dialog box **Image** – An optional image to display at the top of the dialog. For best results, the image should be 462x75 pixels and have a white background. **InfoImage** – Adds an optional image, center and at the bottom of the dialog. This image will appear on top of any information displayed in the dialog if there is any overlap. **ShowBack** – Shows a back button on the dialog enabling the interactive user to return to the previous GUI based action. **Title** – The title text of the dialog box.	Displays an error dialog box. The actual text shown is from the value of this element and not an attribute. To use HTML tags, ensure that you use a <![CDATA[tag (see the example) so that the XML parser doesn't freak out.
ExternalCall	**MaxRunTime** – The maximum allowed run time in seconds for the command. If the command does not finish in this amount time, the process will be forcefully terminated by UI++. **Title** – The title of the external command. This is shown in the log as well as the	Runs the specified external command. The command-line run is contained in the value of the element and not an attribute. Enclosed this command in <![CDATA[XML tags if necessary to prevent the XML parser from mis-interpreting characters in the specified command-line.

[9] A complete list of all default values is listed in the "Default Values" section.

	progress bar but is not used to actually execute the command-line.	
FileRead	**DeleteLine** – Whether the line should be deleted after it is read from the file. Default is **True.** **FileName*** – The full path including the file name of the text file to read from. **Variable*** – The variable name to store the resulting text in.	Reads the first line from the specified file, places the value into the specified variable, and optionally deletes the line from the file.
Info	**Name** – The name of the dialog box **Image** – An optional image to display at the top of the dialog. For best results, the image should be 462x75 pixels and have a white background. **InfoImage** – Adds an optional image, center and at the bottom of the dialog. This image will appear on top of any information displayed in the dialog if there is any overlap. **ShowBack** – Shows a back button on the dialog enabling the interactive user to return to the previous GUI based action. Default value is false. **ShowCancel** – Shows a cancel button allowing the user to gracefully exit UI++ without proceeding. Default is false. **Title** – The title text of the dialog box. 2.9.0.0 **Timeout** – The number of seconds after which the dialog will be automatically dismissed. Default is 0 which results in the dialog not timing out at all. 2.9.0.0 **TimeoutAction** – The action to take when the dialog timeout elapses. Default is "Continue". Other possible values include	Displays an info dialog box. The actual text shown is from the value of this element and not an attribute. To use HTML tags, ensure that you use a <![CDATA[tag (see the example) so that the XML parser doesn't freak out. Both the **Image** and **InfoImage** attributes can specify local images or those retrievable using an HTTP based URL; e.g., http://home.configmgrftw.com/images/coretechnew.png. Simply specify the URL instead of a local path for either of these attributes.

2.9.0.0	"Cancel" or a custom return code which also cancels the dialog and UI++. **TimeoutMsg** – The message to prefix the time out countdown with. Default is "Timeout in".	
Input 2.9.4.0	**Name** – The name of the dialog box **ShowBack** – Shows a back button on the dialog enabling the interactive user to return to the previous GUI based action. Default value is false. **ShowCancel** – Shows a cancel button allowing the user to gracefully exit UI++ without proceeding. Default is false. **Size** – The height of the input dialog displayed. Valid Options include Regular and Tall. Default is Regular. A regular sized Input dialog will display up to 3 controls and a tall will display up to 6. **Title** – The title text of the dialog box.	Displays a dialog with a series of customizable text input boxes, combo/choice boxes, and info text.
Preflight 2.9.4.0	**ShowBack** – Shows a back button on the dialog enabling the interactive user to return to the previous GUI based action. Default value is false. **ShowCancel** – Shows a cancel button allowing the user to gracefully exit UI++ without proceeding. Default is false. **Title** – The title of the preflight dialog. **ShowOnFailureOnly** – Shows the preflight dialog if and only if one or more of the Checks defined for it have failed. Default value is false. **Size** – The height of the preflight dialog displayed. Valid Options include Regular and Tall. Default is Regular.	Displays a dialog containing "preflight" checks. All checks must pass for the dialog to be able to be dismissed successfully. If any checks do not pass, dismissing the dialog results in an error code and a halt to UI++.

RegRead	**Default** – A default value to populate the specified variable with if the registry read operation cannot read a value or returns a blank value. **Hive** – The registry hive to read the value from. Valid values include HKLM and HKCU. HKLM is the default if not specified. **Key*** – The full path to the registry key to read from. **Reg64** – Read from the 64-bit registry instead of the 32-bit registry if using the 32-bit version. Default is True. **Variable*** – The variable name to store the registry value in. **Value*** – The name of the registry value to retrieve.	Reads a value from the registry and places it into a variable.
RegWrite	**Hive** – The registry hive to read the value from. Valid values include HKLM and HKCU. HKLM is the default if not specified. **Key*** – The full path to the registry key to read from. **Reg64** – Write to the 64-bit registry instead of the 32-bit registry if using the 32-bit version. Default is True. **Type** – The registry value type to be written. If not specified, REG_SZ will be used. Valid values include any valid registry value type including REG_SZ and REG_DWORD. **Value*** – The name of the registry value to retrieve.	Write values to the registry in the specified location. If the value already exists, it will be overwritten.
2.10.3.0 **SaveItems**	**Items*** – A comma-separated list of items to save. **Path*** – The path to save the items to.	Saves items to the specified location. Valid values for **Items** include the following: • SMSTSLog

		• UILog • TSVariables • <filename> • <path>\<filename> By default, a value of TSVariables will use the file name of "UI++ Variable Dump <time and date>.txt". To specify a custom file name, append the desired name after specifying TSVariables but separated by a colon; e.g, *TSVariables:vars.txt* will name the file *vars.txt*. Environment variables are valid for the <filename> and <path>\<filename> values as well as the **Path** attribute. Wildcards are valid for the <filename> and <path>\<filename> values.
SoftwareDiscovery	N/A	Searches the registry for installed software.
Switch	**DontEval** – If set to True, do not pass the value of the **OnValue** attribute to VBScript for processing. Default is false. **OnValue*** – The value to match against. Can be a static value, a task sequence variable or a VBScript expression.	This action matches the value specified in the **OnValue** attribute against a series of regular expression cases. If any of the cases match, then the Variable sub-elements are used to set the value of one or more task sequence variables.
TSVar	**Name** or **Variable*** – The variable name to set a value for. (Variable and Name are synonyms. Variable was added for attribute naming consistency with other actions and should be preferred. If both are set in an action, the value of the Variable attribute takes precedence to set the name of the variable). **DontEval** – If set to True, do not pass the value of the element to VBScript for processing. Default is false.	This action statically sets a variable; the variable could be any pre-existing variable or a new one. The actual value set is from the value of this element and not an attribute.

2.9.2.0

TSVarList	**ApplicationVariableBase** -- The base variable to populate with included applications. There is no default value. **PackageVariableBase** – The base variable to populate with included packages. There is no default value.	Creates a variable list populated with application or package information for use with an Install Application task or an Install Package task respectively. One or both of the attributes must have a value for this Action to be processed.
UserAuth 2.10.3.0 2.9.3.0	**Attributes** – A list of valid AD attributes to gather from the user that successfully authenticates using this action. **DisableCancel** – Initially disables the cancel button on the UserAuth dialog. If set to true, the button will be re-enabled if the maximum number of retries without a successfully authentication occurs. Default is false. **Domain** – The default domain to authenticate the user against. This should be in the form of an FQDN. **Group** – A semi-colon separated list of domain security groups; the user must be a member of one of the listed groups in order to pass this Action. **Title** – The title of the user authentication dialog. **MaxRetryCount** – The maximum number of authentication attempts allowed. **ShowBack** – Shows a back button on the dialog enabling the interactive user to return to the previous GUI based action.	Displays a dialog with three partially customizable fields: User name, Password, and Domain.
Vars	**Direction** – Specifies whether to load or save the variables from or to a data file. Valid values are "Load" and "Save"; default value is "Save".	Loads variables from a data file or saves the current variables to data file. Saving only works if UI++ is executed outside of a task sequence; loading works within a task sequence or outside of it. Read-only variables (those beginning with an underscore) and variables beginning with an 'X' are never saved or loaded.

	Filename – The filename to load data from or save data to. Defaults to %temp%\ui++vars.dat.	
WMIRead 2.9.2.0	**Class*** – The WMI Class to read from. **Default** – A default value to populate the specified variable with if the WMI query operation cannot read a value or returns a blank value. **KeyQualifier** – Specifies the WMI class qualifier using the key attribute of the class to identify a specific instance of a class. **Namespace** – The WMI namespace to read from. Root\cimv2 is the default if none namespace is specified. **Property*** – The WMI Property to pull the value from. **Variable*** – The variable name to store the WMI value in. **Query*** – A valid WQL query.	Reads a value from WMI and places it into a variable. There are two ways to retrieve info: by class instance or by query. For class instance, you must specify the Class and Property. If multiple instances of a class exist, then the first will be used unless a key qualifier is specified. If a value is an array, the first value in the array is returned. For a query, you must specify a Query and a Property. Ensure that the query returns the property specified. If the query returns multiple instances, then the first will be used.
WMIWrite	**Class*** – The WMI Class to read from. **Namespace*** – The WMI namespace to write to. Root\cimv2 is the default if none namespace is specified.	Opens or creates a namespace, a WMI class, and a WMI object instance within that namespace. Use a child Property element to write attributes into the opened or newly created object instance. If the class already exists then the properties specified must match those from that class's definition. If the value of key property is the same, the instance will be overwritten; if the values differs, then a new instance will be created.

11 CHANGE LOG HISTORY

The following is the change log prior to version 2.10.0.0.

2.9.6.0

- Fixed
  - **Preflight** action bug; when showing the cancel button explicitly, the all checks passed message would always be shown even if all checks did not pass.

2.9.5.0

- Updated
  - The **Switch** action to be able to set one or multiple task sequence variables per **Case** element using **Variable** sub-elements. The old format introduced in 2.9.2.0 is no longer valid and should be updated in any existing XML configuration files.
- Fixed
  - The **WMIRead** action so that a specified default value would correctly be used if the value in WMI is blank.

2.9.4.0

- Updated
  - **Preflight**, **Input**, and **AppTree** actions so that they can have a cancel button shown by setting the **ShowCancel** attribute to True.

2.9.3.0

- Added
  - Ability to specify a list of domains for the UserAuth action.
  - Ability to disable the cancel button on the UserAuth action.
- Fixed
  - Bug where hitting enter or escape during a **DefaultValues** action where the progress bar is shown would cause a crash.

2.9.2.0

- Added
  - Ability to use a query with the **WMIRead** action.
  - Wired NIC detection during the **DefaultValues** actions.
  - A new **Switch** action similar to the SQL CASE statement and switch statements found in many programming languages.
  - Ability to populate the choices and values in a **ChoiceInput** from a comma-separated list of values.

2.9.1.1

- Fixed

- Bug with the /retry option causing it to be ignored.

2.9.1.0

- Added
  - **ChoiceInput** option to enable or disable sorting of choices within the drop-down list box.
  - A flat dialog mode that gets rid of the rounded dialog corners and the border.
- Fixed
  - Memory leak when gathering usernames in the **DefaultValues** action.

2.9.0.0

- Added
  - Extended description for **Check** items in Preflight actions that show as tooltips when hovering over the **Check** text.
  - Extended failure description **Check** items in Preflight actions that show as tooltips when hovering over the failure icon.
  - **Info** dialog action timeout.
  - **DefaultValues** detects pending reboots based upon one of four different conditions.
  - **DefaultValues** discovers the current user's display name (from AD or AzureAD) if available and the SAM Account name if available.
  - /retry and /fallback command-line switches to specify the number of times that UI++ will attempt to redownload a configuration file from a web location and a fallback configuration file to use in case the download fails.
- Fixed
  - Option to show **Preflight** action only if a check failed skipped action properly but did not initiate next action.
  - First time tooltip for the first **TextInput** field no longer shows in a random location on the screen.
- Updated
  - Tooltips for **TextInput** fields now show the hint text as a title along with an icon.
  - The **DefaultValues** action now recognizes ChassisTypes with code 30, 31, 32 as laptops.
  - Internal message string organization.
  - Log message formatting for some log messages.
  - The library used to download the configuration file from an http location to the same library used to download images from an http location. Both http and https are now supported.

2.8.5.0

- Added
  - Auto-completion for user **ChoiceInput** items in Input actions.
  - Option to only show **Preflight** action dialog if a check fails.
  - Option to use a tall **Preflight** action dialog to show more preflight checks.
  - **DefaultValues** now discovers whether a system is booted using UEFI and if SecureBoot is enabled.
- Updated

- **DefaultValues** action properly collects OS info on Windows 10; specifically, XOSVersion, XOSBuild, and XOSServicePack now collect info from WMI instead of the deprecated Windows API for this.

2.8.2.1

- Fixed
  - Bug in **UserAuth** action that prevented groups from being checked and the XAUthenticatedUser and XAUthenticatedUserDomain variables from being populated.

2.8.2.0

- Added
  - Ability to load **Info** action images from an HTTP location.
- Updated
  - Small code updates to clean up after failed authentication attempts.

2.8.1.0

- Added
  - Added a do not evaluate option to the **TSVar** action to prevent the value being set from VBScript processing.
- Updated
  - The **DefaultValues** action now gathers IP Addresses, IP Subnet Masks, Default IP Gateway, and MAC Addresses.
  - Changed the method that clears memory of user name and password after an authentication attempt in a **UserAuth** action.
  - The TSVar action now accepts either Name or Variable as an attribute to name the task sequence variable being set. This was done for naming consistency. Variable should be preferred and if both are set, Variable takes precedence.

2.8.0.0

- Added:
  - A new **FileRead** action that reads the first line from a specified text file, places the line's value into a specified task sequence variable, and then optionally deletes the line from the text file.
  - A new **ErrorInfo** action that displays a final error message to the interactive user along with a cancel button that exits UI++.
  - A new **ExternalCall** action that calls an external command-line.

2.7.3.0

- Added:
  - Ability to add a static image to the **Info** action dialog.
- Fixed:
  - Conditions to individual **Preflight Check** items now work.

- Properly set the first input control on **Input** action dialogs to the first item even when all items are valid and the OK button is enabled.
- Modified banner image display on the **Info** action dialog so that the background is transparent when using a smaller image.
- Issues introduced with adding the progress bar to the **DefaultValues** action.

2.7.2.2

- Fixed:
  - Issues introduced with adding the progress bar to the **DefaultValues** action.

2.7.2.0

- Added:
  - A checkbox item on an **Input** action.
  - An optional progress bar for the **DefaultValues** action with customizable progress text.
- Fixed:
  - Bug where not all input items on an **Input** action dialog would trigger validation. This would prevent the OK button from being enabled even though all user input was valid.
  - The **SoftwareListRef** element is properly used instead of the **SoftwareRef** element for **TSVarList** Actions.
  - Fixed collection method for the processor architecture for the **DefaultValues** action.
  - Fixed a bug that caused a crash if more inputs are added to an **Input** action dialog than it can display.

2.7.1.0

- Added:
  - The ability to add a back button to all UI actions.
  - Ability to disable the domain field on a **UserAuth** action.

2.7.0.1

Fixed:

- Loading variables previously saved using a Vars action outside of a task sequence were not loaded using a Vars Action during a task sequence.

2.7.0.0

- Added:
  - Boolean Variable Negation
  - Software Discovery
  - Task Sequence Variable List creation
  - Default values for **WMIRead** and **RegRead** actions
  - Optional branding banner image to Info action dialogs
  - Default value creation for TPM state
  - Ability to retrieve a subset of the default values
  - Default Value specifying the virtual machine type

- Fixed:
  - Multiple bugs when using UI++ as a pre-start command.
  - Minor WMI read and write updates to properly release resources.
- Updated:
  - Reduced the size of the drop-list for combo-boxes in Input Actions to a maximum five shown values.
  - All icons

2.4.5.5

- Added:
  - Command-line switch to disable use of the Task Sequence variable editor or dumping the variables to a file.
- Fixed:
  - The OK button on the pre-flight dialog was not default and thus did not respond to pressing the Enter key.
  - WMI Write operation bug.

2.4.5.0

- Updated:
  - **User Info** dialog to break long lines if needed on commas, periods, forward slashes and backslashes.
  - Default regular expression used for the username field adjusted to allow additional characters per https://msdn.microsoft.com/en-us/library/bb726984.aspx.
  - Internal XML parsing library, PugiXML to version 1.6.
  - Default value for text input and choice input on **Input** action dialogs is automatically set to a previous value for the input elements variable if one exists.
- Added
  - Ability to customize the prompt, hint, and question as well as the regular expression used to validate the input in the username, password, and domain fields of a **UserAuth** dialog.
  - New action to save and load variables and their values to or from a data file. This allows running UI++ as a program that a Task Sequence depends upon to collect user information in the full OS, save the collected info, and then load it for use during the task sequence.

2.4.0.0

- Bugfixes:
  - Package variables now correctly have three digits instead of two
  - Corrected the **UserAuth** action so that group members are properly returned even if the group check option is not used
- Added:
  - Ability to read from and write to the 64-bit registry if using the 32-bit version
  - Membership check in one of a list of groups instead of just a single group
  - Set **AppTree** to not expanded by default
  - Cancel button on **UserAuth** action
  - Optional cancel button on **User Info** action

    - Add a software reference to the same software item multiple times in an **AppTree**
    - Set a **TextInput** item to show as a password field hiding user input
    - Version output to log file at the start of running
- Re-enabled:
    - Inclusions in **AppTree** (exclusions are still disabled)

2.3.0.5

Bugfix for running in WinPE on Windows Server without WLAN capabilities.

2.3.0.0

- Bugfixes:
    - Regular expression case insensitivity
    - WinPE detection
- Added:
    - WLAN and SSID detection
    - Current computer name , computer domain, and computer OU detection
    - Authenticated user's group membership detection during the **ADAuth** action
    - Added **Preflight** Action
    - Info fields to the **Input** [dialog] action
    - Ability to change accent color of all dialog boxes
    - Ability to remove icon from top right corner of dialog boxes
    - Common snippet section in this document to provide useful snippets
- Removed:
    - (Temporarily) Inclusions and exclusions from the AppTree action.

2.2.0.0

- Updated and reorganized this documentation a bit.
- Added extended **AppTree** capabilities
    - Default items
    - Required items
    - Hidden items
    - Included items
- On-demand task sequence variable dump to a text file
- Large sizes for the **Input** and **AppTree** action dialogs
- Use of a configuration file from an HTTP URL.

2.1.0.0 (Beta 2) Release

- Added basic **AppTree** functionality
- Added Registry Write capabilities
- Added Required attribute for user input
- Updated icons
- Minor bugfixes
- Included non-debug exes in download

2.0.1.0 Beta Release

- Added WMI Read and Write capabilities
- Added user authentication against AD

2.0.0.1 Private Beta Release

- Initial beta release

12 Known Issues

The following is a list of currently known issues and workarounds in the current version.

Issue	Workaround
When using inclusions, if there is a circular chain of included items and one of the items in this chain is also set as default, all items will be treated as required.	Don't use circular inclusion chains.
Random crashes in WinPE 10 1511 mainly on UEFI systems.	Don't use WinPE 10 1511 – it's buggy and I can't fix that.

Please submit other issues encountered via the contact form at http://blog.configmgrftw.com/contact/.

13 License

This application is released subject to the following license:

Microsoft Public License (Ms-PL)

This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software.

1. Definitions

The terms "reproduce," "reproduction," "derivative works," and "distribution" have the same meaning here as under U.S. copyright law.

A "contribution" is the original software, or any additions or changes to the software.

A "contributor" is any person that distributes its contribution under this license.

"Licensed patents" are a contributor's patent claims that read directly on its contribution.

2. Grant of Rights

(A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create.

(B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software.

3. Conditions and Limitations

(A) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks.

(B) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically.

(C) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software.

(D) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license.

(E) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.